**PRACTICAL NO.:1**

**AIM OF PRACTICAL AS PER LIST:** **To study the basic concepts of Kotlin: data types,**

**variables, operators, strings etc.**

**PROBLEM STATEMENT:**Using the commands of data types variables in the programs with suitable examples.

**THEORY (IF APPLICABLE):** Data Types:

Int: Represents whole numbers (e.g., 1, 10, -5).

Double: Represents decimal numbers (e.g., 3.14, -0.5).

Boolean: Represents true or false values (e.g., true, false).

Char: Represents a single character (e.g., 'A', '2').

String: Represents a sequence of characters (e.g., "Hello, World!").

Variables:In Kotlin, you can declare variables using val (immutable) or var (mutable)**:**

val name: String = "John" // Immutable variable

var age: Int = 30 // Mutable variable

Operators:

Arithmetic Operators: + (addition), - (subtraction), \* (multiplication), / (division), % (modulo).

Comparison Operators: == (equals), != (not equals), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to).

Logical Operators: && (and), || (or), ! (not).

Assignment Operators: = (assignment), += (add and assign), -= (subtract and assign), and so on.

Strings:String Concatenation:

val firstName = "John"

val lastName = "Doe"

val fullName = firstName + " " + lastName

String Interpolation: val name = "Alice"

val age = 25

val message = "My name is $name and I am $age years old."

String Functions:Kotlin provides various string functions like length, substring, startsWith, endsWith, toUpperCase, toLowerCase, and more for working with strings.

Type Conversion:

You can convert between different data types using methods like toInt(), toDouble(), toString(), etc. For example:

val numberString = "42"

val number = numberString.toInt()

Null Safety:

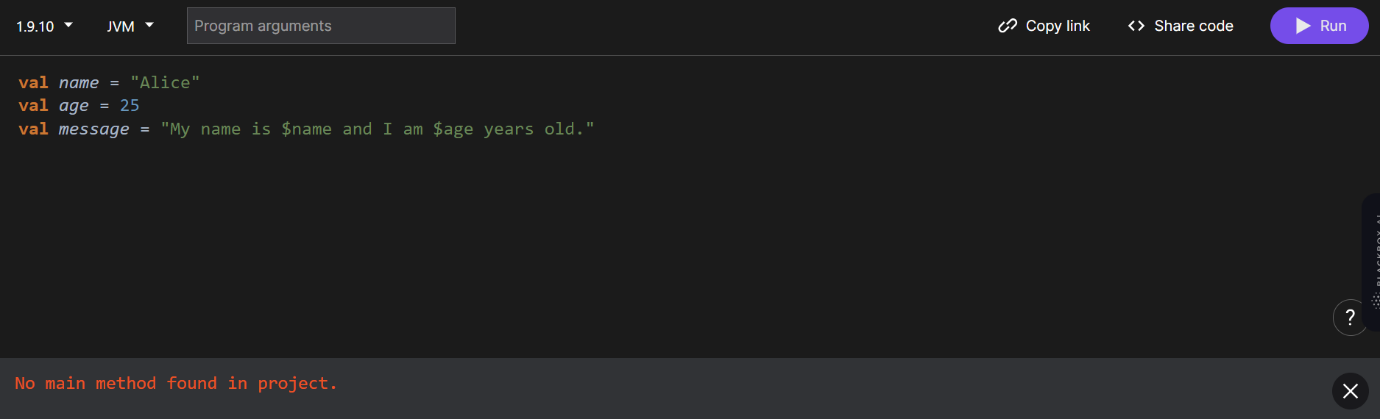
Kotlin provides null safety with nullable and non-nullable types. To declare a nullable variable, you use Type?. For example:

val nullableString: String? = null

**PROGRAM:** **val name = "Alice"**

**val age = 25**

**val message = "My name is $name and I am $age years old."**

**OUTPUT SCRRENSHOT:**

**CONCLUSION:**Therefore I have studied the basic syntax of kotlin data types declaring variables.

**PRACTICAL NO.:2**

**AIM OF PRACTICAL AS PER LIST:** **Kotlin programs based on Loops , arrays &amp; ranges**

**PROBLEM STATEMENT:Program to find maximum element using a for each loop.**

**THEORY (IF APPLICABLE):** In Kotlin, a loop is a control flow statement that allows repeated execution of a block of code for a certain number of times or until a certain condition is met. Kotlin provides several types of loops, including:

For loop: It is used to iterate over a range of values or an iterable collection. To create a range of values, Kotlin provides the .rangeTo() and .rangeUntil() functions from the kotlin.ranges package. The for loop can also be used to iterate over arrays and other collections of elements

While loop: It repeatedly executes a block of code as long as a certain condition is true. The while loop in Kotlin is similar to a do-while loop, but the code within the do block is executed before the condition is checked

Do-while loop: It is similar to the while loop, but guarantees that the block of code will be executed at least once before the condition is checked

To iterate numbers in reverse order, Kotlin provides the downTo function instead of using a range with a negative step value. It is also possible to iterate over numbers with an arbitrary step (not necessarily 1) using the step function

Ranges in Kotlin consist of a start, a stop, and the step. The start and stop are inclusive in the range, and the value of the step is by default 1. There are three ways to create a range in Kotlin:

Using the (..) operator

Using the rangeTo() function

Using the downTo() function

To define a custom progression step, use the step function on a range. The last element of the progression is calculated this way: For a positive step, the maximum value not greater than the end value such that (last - first) % step == 0 is taken as the last element. For a negative step, the minimum value not less than the end value such that (last - first) % step == 0 is taken as the last element

To step through a fixed range in a loop in Kotlin where the step value changes on every iteration, you can use a while loop and manually increment the loop variable by the desired step value

**PROGRAM:** **fun main() {**

**val numbers = intArrayOf(10, 5, 15, 8, 20)**

**var max = Int.MIN\_VALUE**

**numbers.forEach {**

**if (it > max) {**

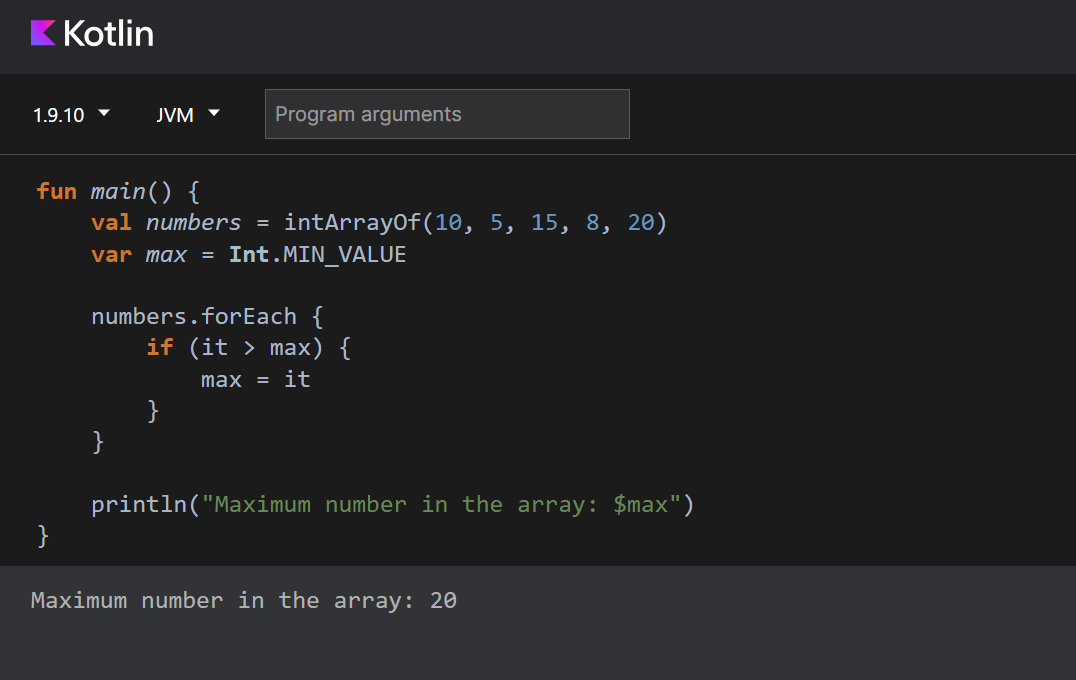
**max = it**

**}**

**}**

**println("Maximum number in the array: $max")**

**}**

**OUTPUT SCRRENSHOT:** ****

**CONCLUSION:Therefore I have implemented the loops arrays and ranges.**

**PRACTICAL NO.:3**

**AIM OF PRACTICAL AS PER LIST:** **Kotlin programs based on functions.**

**PROBLEM STATEMENT:** Program for Recursive Function to Calculate Factorial

**THEORY (IF APPLICABLE):** In Kotlin, functions are a fundamental building block of the language, and they offer a powerful way to structure and organize code. Here's a summary of key points regarding functions in Kotlin:

Function Declaration:

Functions are declared using the fun keyword, followed by the function name, parameter list, return type, and function body.

Example: fun add(a: Int, b: Int): Int { return a + b }

Parameters:

Functions can accept zero or more parameters, and each parameter has a name and a type.

Example: fun greet(name: String, message: String) { /\* ... \*/ }

Return Types:

Functions may have a return type, which specifies the type of value the function will return.

Example: fun add(a: Int, b: Int): Int { return a + b }

Function Calls:

Functions are called by providing arguments that match the parameter list.

Example: val result = add(5, 3)

Default Parameter Values:

You can specify default values for function parameters, making them optional in function calls.

Example: fun greet(name: String, message: String = "Hello") { /\* ... \*/ }

Named Arguments:

In function calls, you can use named arguments to specify which parameter is receiving which value.

Example: greet(message = "Hi", name = "Alice")

Single-Expression Functions:

If a function consists of a single expression, you can use the simplified syntax.

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

Infix Functions:

You can define infix functions with the infix keyword, which allows for more readable function calls.

Example: infix fun Int.plusFive() = this + 5

Function Overloading:

Kotlin supports function overloading, allowing you to define multiple functions with the same name but different parameter lists.

Example: fun greet(name: String) { /\* ... \*/ } and fun greet(name: String, message: String) { /\* ... \*/ }

Higher-Order Functions:

Kotlin supports higher-order functions, which are functions that can take other functions as parameters or return functions.

Example: fun operate(a: Int, b: Int, operation: (Int, Int) -> Int): Int { return operation(a, b) }

Lambda Expressions:

Lambdas are concise ways to define small, anonymous functions, often used as arguments to higher-order functions.

Example: (a, b) -> a + b

Extension Functions:

Kotlin allows you to add new functions to existing classes without modifying their source code, known as extension functions.

Example: fun String.isPalindrome() { /\* ... \*/ }

Recursion:

Functions can be recursive, meaning they call themselves, allowing for solving problems by dividing them into smaller, similar subproblems.

Example: fun factorial(n: Int): Int = if (n == 0) 1 else n \* factorial(n - 1)

Functions in Kotlin are versatile and provide a flexible way to structure code, support code reuse, and make code more readable and maintainable. They are a fundamental part of Kotlin's expressive and concise syntax.

**PROGRAM:** **fun factorial(n: Int): Int {**

**return if (n == 0) 1 else n \* factorial(n - 1)**

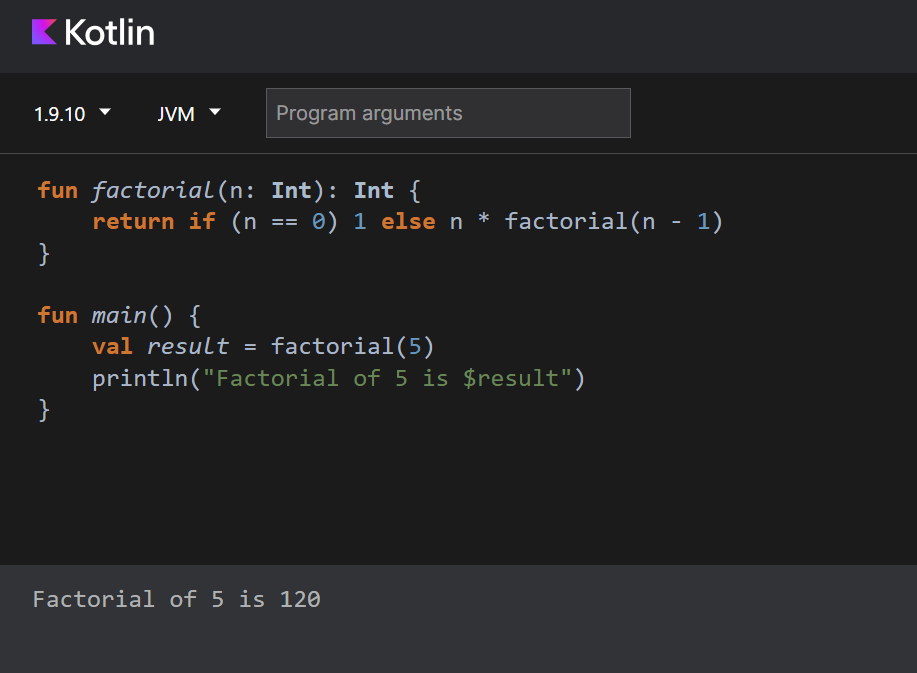
**}**

**fun main() {**

**val result = factorial(5)**

**println("Factorial of 5 is $result")**

**}**

**OUTPUT SCRRENSHOT:** ****

**CONCLUSION:Here I have completed kotlin program in functions.**

**PRACTICAL NO.:4**

**AIM OF PRACTICAL AS PER LIST:** **Kotlin programs based on inheritance**

**PROBLEM STATEMENT:Program for basic inheritance of Animals.**

**THEORY (IF APPLICABLE):** Inheritance is a fundamental concept in object-oriented programming that allows you to create new classes (subclasses or derived classes) based on existing classes (superclasses or base classes). In Kotlin, inheritance is a key feature that enables code reuse and promotes the building of hierarchical relationships between classes. Here's a brief overview of inheritance in Kotlin:

Superclass (Base Class):

The superclass, also known as the base class, is the class that is being extended.

It serves as the blueprint for creating new classes.

Subclass (Derived Class):

The subclass, also known as the derived class, inherits the properties and behaviors of the superclass.

It can add new properties or behaviors and can also override the superclass's methods.

open Modifier:

In Kotlin, classes are declared as final by default, meaning they cannot be inherited from.

To allow a class to be a superclass, you need to mark it with the open keyword.

open class Animal {

// ...

}

override Keyword:

In a subclass, you can use the override keyword to redefine a method from the superclass, providing a new implementation.

override fun someMethod() {

//

**PROGRAM:** **open class Animal(val name: String) {**

**fun speak() {**

**println("$name makes a sound")**

**}**

**}**

**class Dog(name: String) : Animal(name) {**

**fun bark() {**

**println("$name barks")**

**}**

**}**

**class Cat(name: String) : Animal(name) {**

**fun meow() {**

**println("$name meows")**

**}**

**}**

**fun main() {**

**val dog = Dog("Buddy")**

**val cat = Cat("Whiskers")**

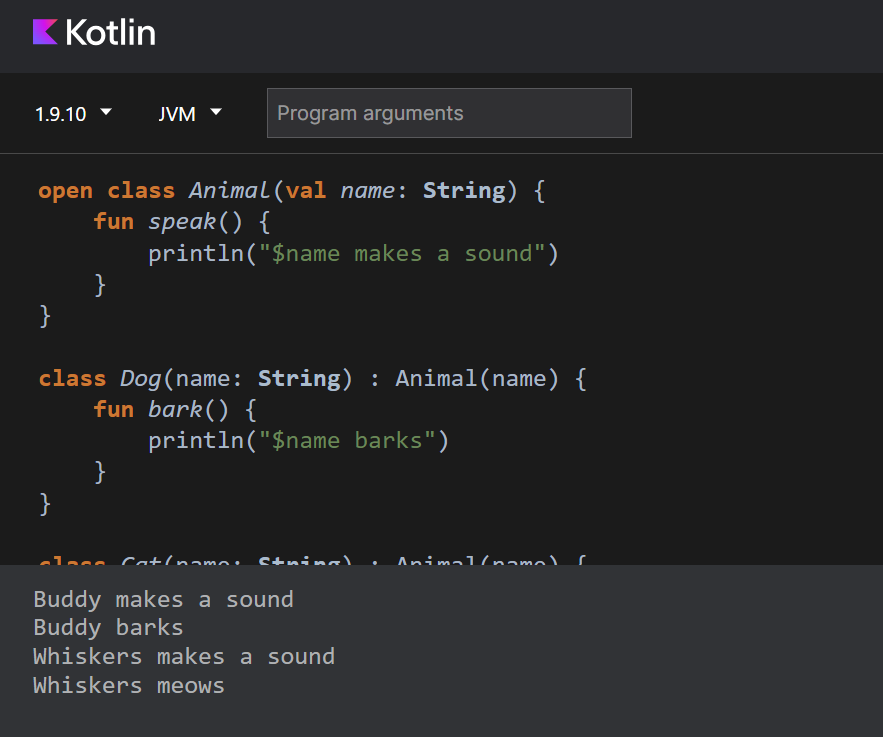
**dog.speak()**

**dog.bark()**

**cat.speak()**

**cat.meow()**

**}**

**OUTPUT SCRRENSHOT:** ****

**CONCLUSION:Here I have implemented kotlin program to show inheritance in animals.**

**PRACTICAL NO.:5**

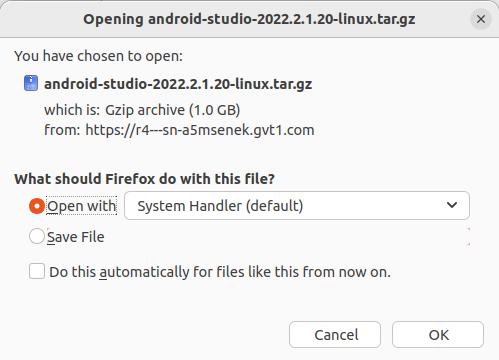
**AIM OF PRACTICAL AS PER LIST:** **To set up an Android studio &amp; create birthday card App**

**PROBLEM STATEMENT:** **To set up an Android studio & create birthday card App**

## THEORY (IF APPLICABLE): Download Android Studio

1. Open any web browser and navigate to the [Android Studio download page](https://developer.android.com/studio/?gclid=Cj0KCQiAjJOQBhCkARIsAEKMtO3zEhdK4_I0CEZic3UH4dl-9gVXuHFR9dCl3TOHKjmv3xWLU3UxfhYaApfAEALw_wcB&gclsrc=aw.ds).
2. This is the Android Developers website, where you can download Android Studio. This page automatically detects your operating system. Click **Download Android Studio**. The **Terms and Conditions** page with the **Android Studio License Agreement** opens.
3. Read the **License Agreement**.
4. At the bottom of the page, if you agree with the terms and conditions, select the **I have read and agree with the above terms and conditions** checkbox.
5. Click **Download Android Studio** to start the download.
6. When prompted, save the file to a location where you can easily locate it, such as the Downloads folder.
7. Wait for the download to complete. This may take a while and may be a good moment to enjoy some tea!

Install Android Studio on Linux



Open the Downloads folder in the terminal.

1. Extract the archive with the tar command.

tar -xzvf android-studio-2022.2.1.20-linux.tar.gz

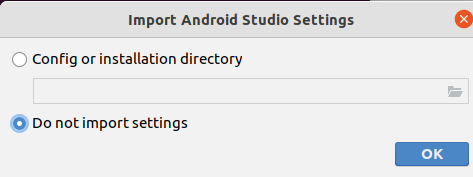
1. Navigate to the android-studio/bin directory.

cd android-studio/bin

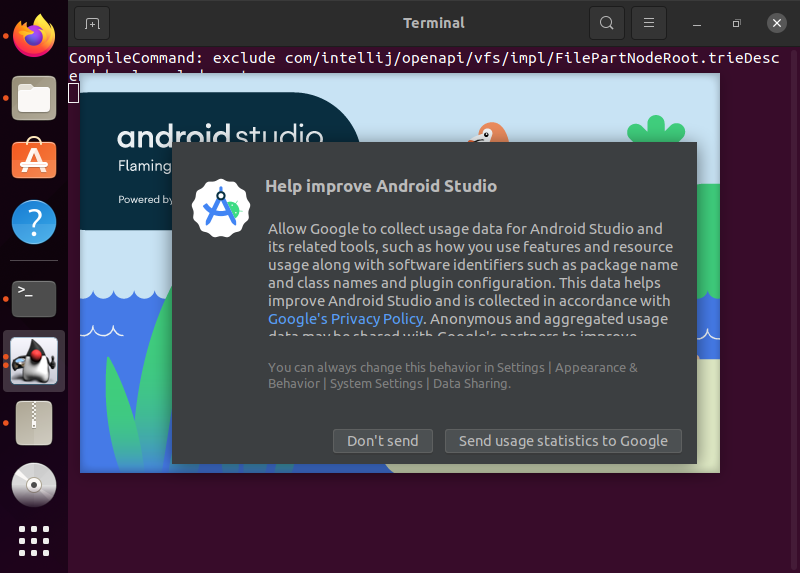
1. Run studio.sh

./studio.sh

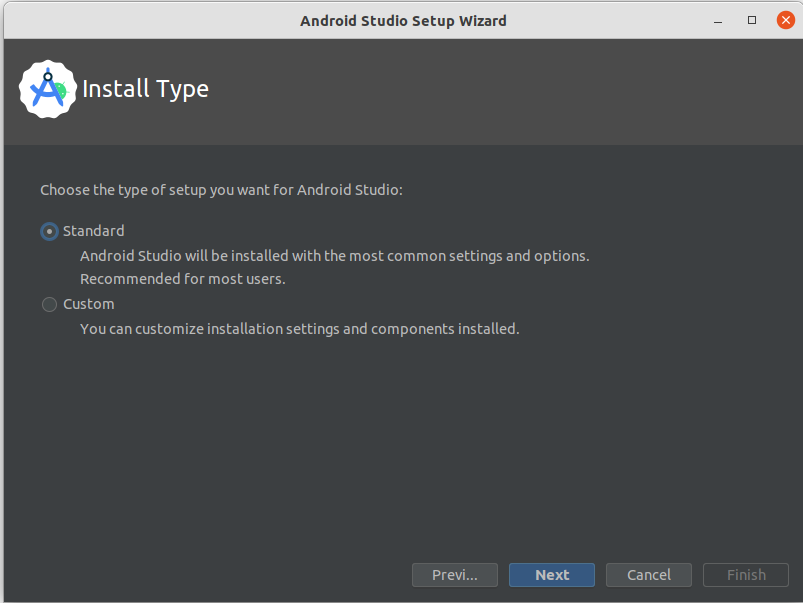
1. Keep **Do not import settings** selected and click **OK** on the prompt.



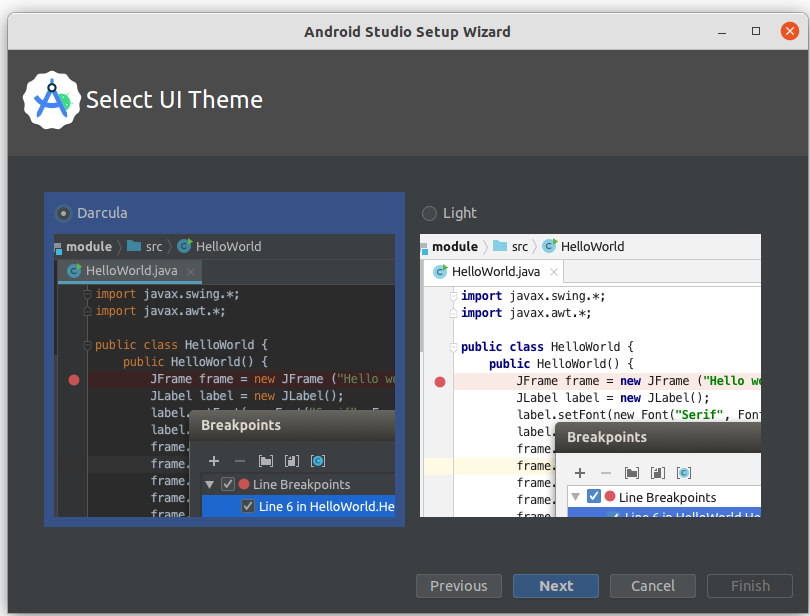
1. Choose whether or not to share usage data with Google.



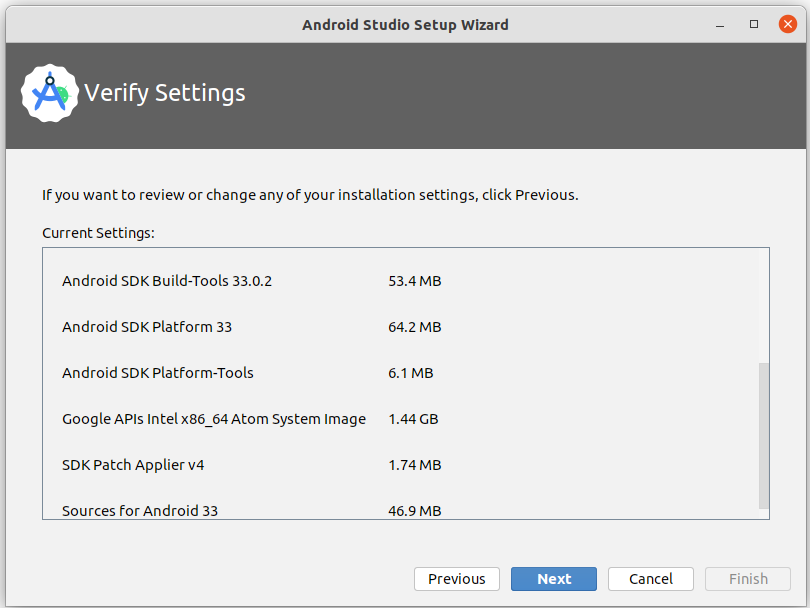
1. Keep **Standard** as the selected install type. Click **Next** to continue.



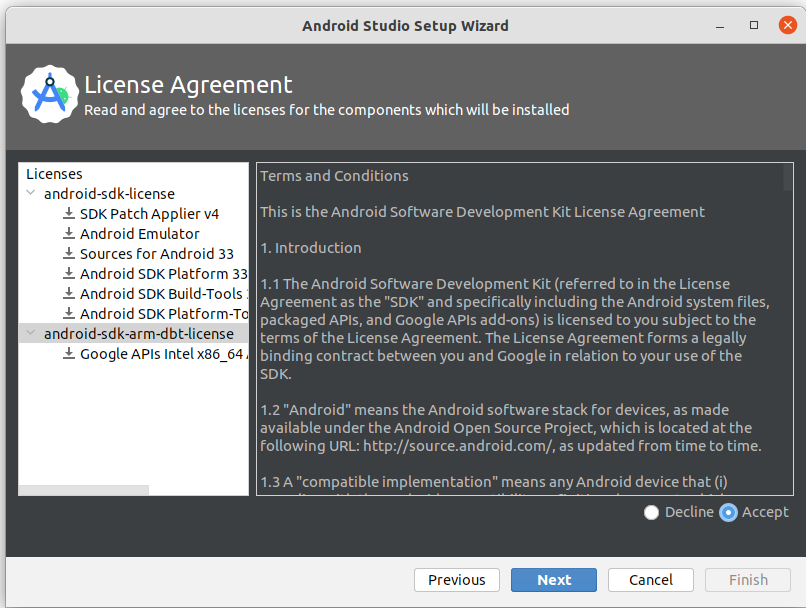
1. Choose your preference of light or dark theme. Screenshots in this course use the light theme, but choose whichever one you prefer. You can always change this later.



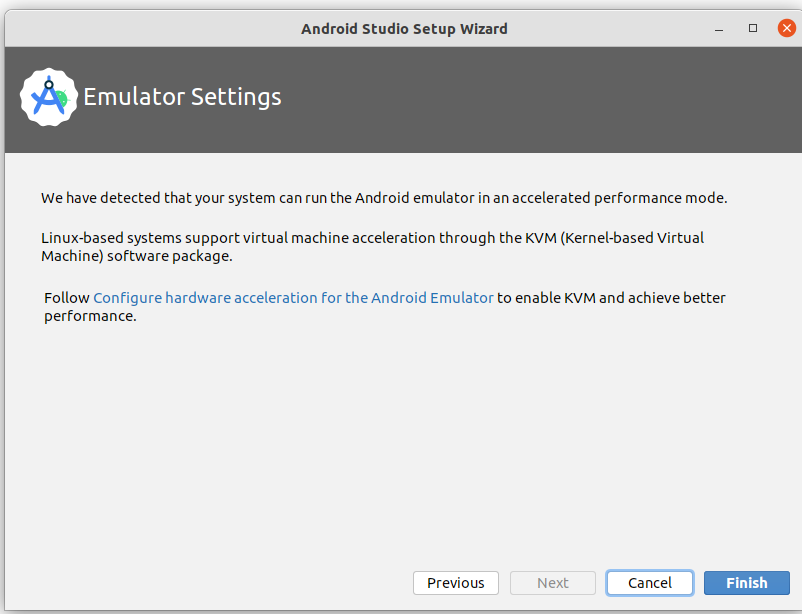
1. Accept all the default settings and click **Next**.



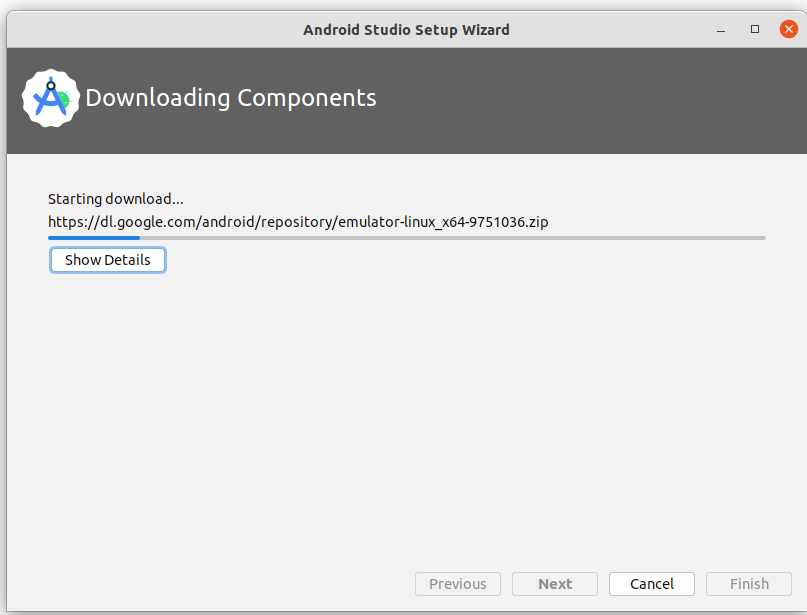
1. Read and agree to the **License Agreement** for the Android SDK and Android NDK, and click **Next**.



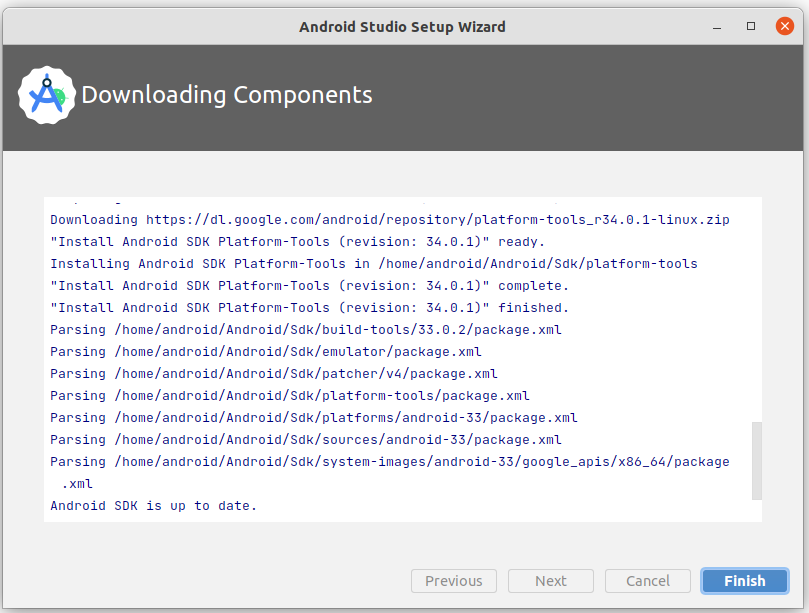
1. You may also see some additional information about hardware acceleration and the Android emulator. Click **Finish**.



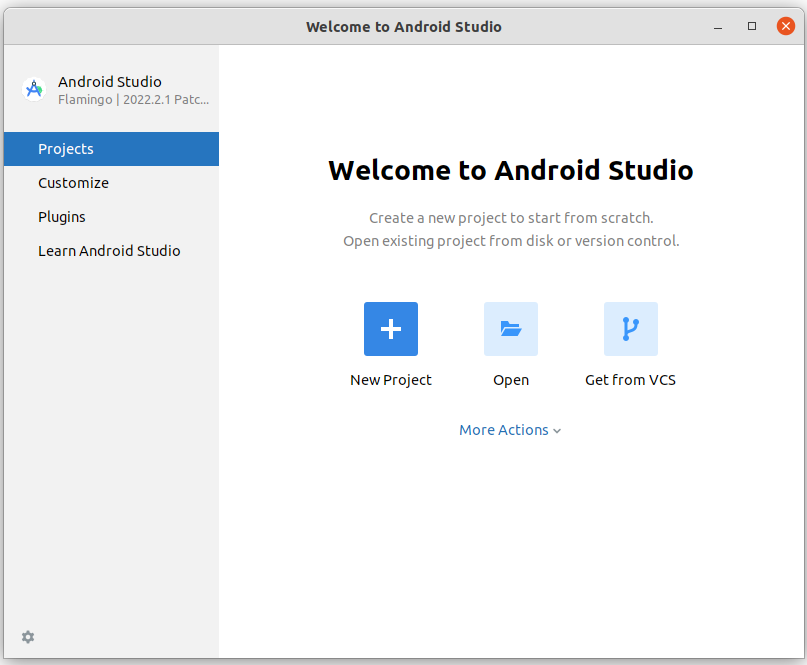
1. During the installation, the setup wizard downloads and installs additional components and tools needed for Android app development.



When the installation is complete, click **Finish**.



1. The **Welcome to Android Studio** dialog displays and you're ready to start creating apps!



**PROGRAM:** package com.example.happybirthday

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.padding

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Surface

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.tooling.preview.Preview

import com.example.happybirthday.ui.theme.HappyBirthdayTheme

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

HappyBirthdayTheme {

*// A surface container using the 'background' color from the theme*

Surface(

modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

Greeting("BIRTHDAY")

}

}

}

}

}

@Composable

fun Greeting(name: String, modifier: Modifier = Modifier) {

Text(

text = "HAPPY $name!",

modifier = Modifier

)

}

@Preview(showBackground = true)

@Composable

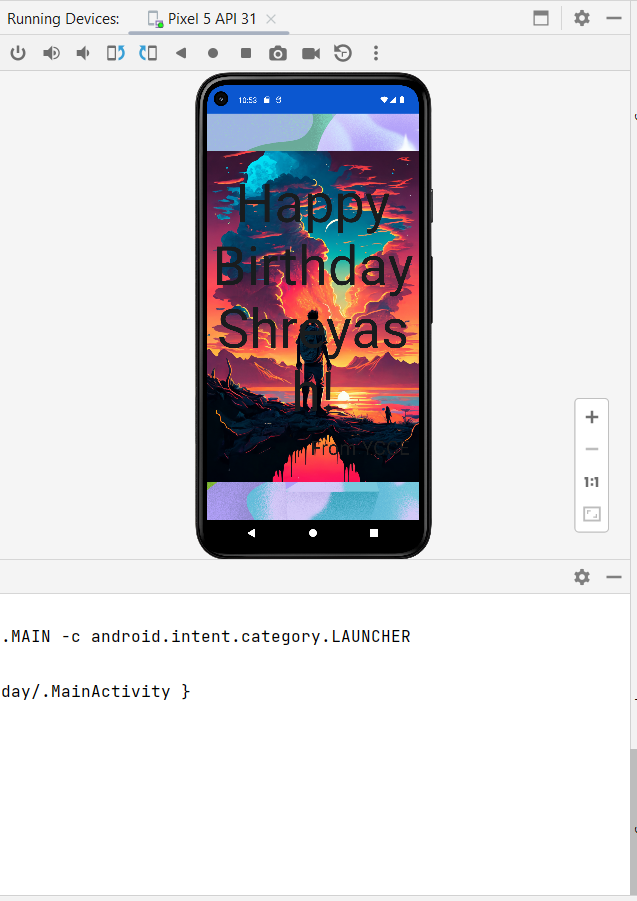
fun GreetingPreview() {

HappyBirthdayTheme {

Greeting("SHREYASH")

}

}

**OUTPUT SCRRENSHOT:** ****

**CONCLUSION:Therefore, I have built the Happy birthday card app Succesfully.**

**PRACTICAL NO.: 6**

**AIM:**

Create a Business card App.

**PROBLEM STATEMENT:**

Create a Business card App.

**PROGRAM:**

package com.example.businesscard

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.material3.Icon

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Surface

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import com.example.businesscard.ui.theme.BusinessCardTheme

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

BusinessCardTheme {

// A surface container using the 'background' color from the theme

Surface(

modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

Greeting()

}

}

}

}

}

@Composable

fun Greeting(modifier: Modifier = Modifier) {

Column(

verticalArrangement = Arrangement.Center,

horizontalAlignment = Alignment.CenterHorizontally

) {

Image(painter = painterResource(R.drawable.android\_logo), contentDescription = null,

Modifier.size(150.dp))

Text(

text = "Shreyash Panage",

fontSize = 30.sp

)

Text(

text = "Android Developer",

color = Color.Green,

fontWeight = FontWeight.Bold,

)

}

Column(

verticalArrangement = Arrangement.Bottom,

horizontalAlignment = Alignment.CenterHorizontally

){

Row {

Icon(painter = painterResource(R.drawable.phoneicon), contentDescription = null,

Modifier.padding(5.dp))

Text(

text = "+91 9172859635",

Modifier.padding(5.dp)

)

}

Row() {

Icon(painter = painterResource(R.drawable.emailicon), contentDescription = null,

Modifier.padding(top = 5.dp, bottom = 10.dp, end = 5.dp))

Text(

text = "Shreyashpanage90@gmail.com",

Modifier.padding(top = 5.dp, bottom = 10.dp, start = 5.dp)

)

}

}

}

@Preview(showBackground = true)

@Composable

fun GreetingPreview() {

BusinessCardTheme {

Greeting()

}

}

**OUTPUT SCRRENSHOT:**

****

**CONCLUSION:** In this practical we built a business card app using jetpack compose and Kotlin and installed it in our mobile phones.

**PRACTICAL NO.: 7**

**AIM:**

Design a calculator App using Kotlin.

**PROBLEM STATEMENT:**

Design a calculator App using Kotlin.

**PROGRAM:**

package com.example.calculator\_compose

import androidx.compose.foundation.background

import androidx.compose.foundation.horizontalScroll

import androidx.compose.foundation.layout.\*

import androidx.compose.material.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Alignment

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.unit.Dp

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import com.example.calculator\_compose.ui.theme.LightGray

import com.example.calculator\_compose.ui.theme.Orange

import org.w3c.dom.Text

import java.lang.reflect.Modifier

@Composable

fun calculator(

state: stateCalculator,

buttonSpacing: Dp = 8.dp,

modifier: androidx.compose.ui.Modifier = androidx.compose.ui.Modifier,

onAction: (actionsCalculator) -> Unit

) {

Box(modifier = modifier) {

Column(

modifier = androidx.compose.ui.Modifier

.fillMaxWidth()

.align(Alignment.BottomCenter),

verticalArrangement = Arrangement.spacedBy(buttonSpacing)

) {

Text(

text = state.numer1 + (state.operation?.sumbol ?: "") + state.number2,

textAlign = TextAlign.End,

modifier = androidx.compose.ui.Modifier

.fillMaxWidth()

.padding(vertical = 32.dp),

fontWeight = FontWeight.Light,

fontSize = 80.sp,

color = Color.White,

maxLines = 2

)

Row(

modifier = androidx.compose.ui.Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.spacedBy(buttonSpacing)

) {

btCalculator(

symbol = "AC",

modifier = androidx.compose.ui.Modifier.background(LightGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Clear)

}

)

btCalculator(

symbol = "\u232B",

modifier = androidx.compose.ui.Modifier.background(LightGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Delete)

}

)

btCalculator(

symbol = "\u0025",

modifier = androidx.compose.ui.Modifier.background(LightGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Operation(operationCalculator.porcentaje))

}

)

btCalculator(

symbol = "\u00F7",

modifier = androidx.compose.ui.Modifier.background(Orange)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Operation(operationCalculator.Divide))

}

)

}

Row(

modifier = androidx.compose.ui.Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.spacedBy(buttonSpacing)

) {

btCalculator(

symbol = "7",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(7))

}

)

btCalculator(

symbol = "8",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(8))

}

)

btCalculator(

symbol = "9",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(9))

}

)

btCalculator(

symbol = "\u00D7",

modifier = androidx.compose.ui.Modifier.background(Orange)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Operation(operationCalculator.Multiply))

}

)

}

Row(

modifier = androidx.compose.ui.Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.spacedBy(buttonSpacing)

) {

btCalculator(

symbol = "4",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(4))

}

)

btCalculator(

symbol = "5",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(5))

}

)

btCalculator(

symbol = "6",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(6))

}

)

btCalculator(

symbol = "\u2212",

modifier = androidx.compose.ui.Modifier.background(Orange)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Operation(operationCalculator.Subtract))

}

)

}

Row(

modifier = androidx.compose.ui.Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.spacedBy(buttonSpacing)

) {

btCalculator(

symbol = "1",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(1))

}

)

btCalculator(

symbol = "2",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(2))

}

)

btCalculator(

symbol = "3",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Number(3)) })

btCalculator(

symbol = "\u002B",

modifier = androidx.compose.ui.Modifier.background(Orange)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Operation(operationCalculator.Add)) }

) }

Row(

modifier = androidx.compose.ui.Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.spacedBy(buttonSpacing)

) {

btCalculator(

symbol = "0",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(2f)

.weight(2f),

onClick = {

onAction(actionsCalculator.Number(0))

},

alignStart = true // Alineación personalizada para el texto del botón "0"

)

btCalculator(

symbol = ".",

modifier = androidx.compose.ui.Modifier.background(Color.DarkGray)

.aspectRatio(1f)

.weight(1f), onClick = {

onAction (actionsCalculator.Decimal) })

btCalculator(

symbol = "\u003D",

modifier = androidx.compose.ui.Modifier.background(Orange)

.aspectRatio(1f)

.weight(1f), onClick = {

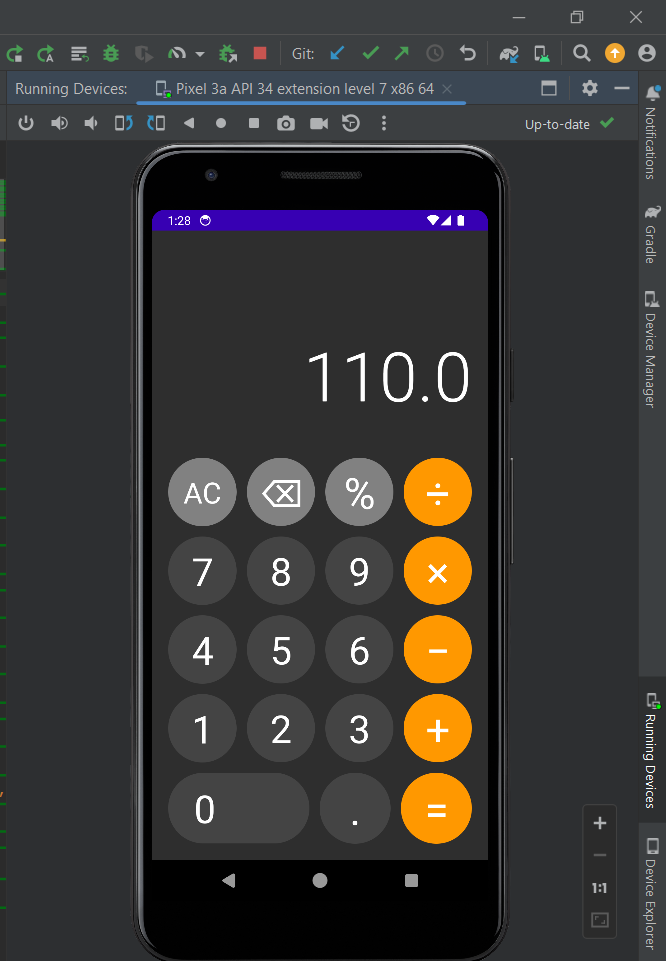
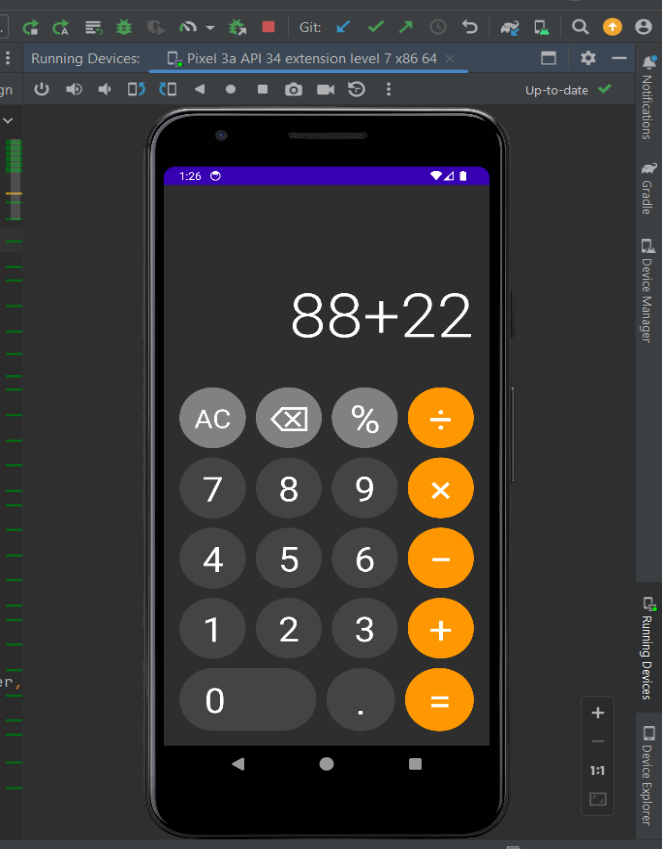
onAction (actionsCalculator.Calculate) })}

}

}

}

**OUTPUT SCRRENSHOT:**



**CONCLUSION:** In this practical we built a calculator card app using jetpack compose and Kotlin and installed it in our mobile phones.