2. Words

Accessibility

In the context of this course, [accessibility](https://developer.android.com/guide/topics/ui/accessibility) refers to the design and functionality of your app, so that it is easier for more people, including those with disabilities, to use. It also means that your app will work well with assistive technologies such as the [Talkback screen reader](https://support.google.com/accessibility/android/answer/6283677?hl=en). Making your app more accessible leads to an overall better user experience, which benefits all your users.

adb (Android Debug Bridge)

**adb** for **Android Debug Bridge** is a versatile command-line tool that lets you communicate with an Android device. You can issue commands to install and debug apps, as well as run a variety of commands on a device.

Android

Android is a mobile operating system designed primarily for touchscreen mobile devices such as smartphones and tablets, but has since expanded to watches, TVs, and cars.

Android app

An **Android app**is a software application that runs on an **Android device**.

Android device

An **Android device**, such as a phone or tablet, is any hardware computing device capable of running the [Android operating system](https://source.android.com/) and **Android apps**.

Android Emulator

**Android Emulator** simulates an Android device on your computer. While a phone may be called a "physical device", an emulator is a "virtual device".

Android Studio

**Android Studio** is the official **IDE** or integrated development environment for the Android operating system, and includes everything you need to create **Android apps**.

Android Virtual Device (AVD)

An **Android Virtual Device**, or **AVD,** is a configuration that defines a specific **Android** phone, tablet, Wear, TV, or Automotive OS device that you want to simulate in an **Android Emulator**.

APK (Android Application Package)

**APK (Android Application Package)** is the **package** file format Android uses to distribute and install mobile apps. When you download an app from Google Play, it automatically downloads and installs the **APK** of the app for you. **Android Studio** can help you create APKs of your apps.

Argument

An **argument** is a value that is passed to **functions**. Arguments can be values, or **variables**, or even other functions.

AVD Manager

The **AVD Manage**r is an interface you can launch from **Android Studio** that helps you create and manage **Android Virtual Devices (AVDs)**.

Boolean

A **Boolean** is a **data type** that has one of two possible values, true and false. Booleans are used with **conditional**statements, letting you change the **control flow** of your program depending on whether the Boolean condition evaluates to true or false.

Button

A **Button** is a type of **View** that displays a piece of text and can be clicked. When touched, the Button can trigger code called a **click handler** that performs an action in response.

Camel case

**Camel case** means to capitalize the beginning of each word and remove all spaces in between. This is a convention for naming **functions**, **variables**, and **classes** in Kotlin, such as LinearLayout, MainActivity, or setOnClickListener.

Class

A **class** is like a blueprint for an **object**. A class defines **properties** (as **variables**) that are common to all objects described in the class, as well as actions (as **functions** called **methods**) for these objects. A class is defined using the class keyword in Kotlin.

A class is similar to how an architect's blueprint plans are not the house; they are the instructions of how to build the house. The house is the actual thing or **object instance** created according to the blueprint. For example, you might have a class PetDog that includes properties for the name and breed. You can then create object instances myPetDog and myFriendsPetDog. These two instances would have different names and breeds, but are both of class PetDog.

Click Listener

A **click listener** is a small piece of code that can be attached to a View. The click listener waits for the view to be tapped, and when the view is tapped, it executes a **click handler**.

A **click handler** is a piece of code that is invoked by a **click listener** when the View is tapped.

Code (Source code)

Computers, including mobile devices, execute lists of instructions called **programs** to carry out their functionality. An Android device is an example of a computer, and an app is an example of a program. Android apps are written in a programming language, such as Kotlin, that can be understood by the device. These instructions are called **code (**or **source code)**.

Code editor

Text editors are used to create or edit plain text files. Code editors are designed for creating or editing **source code**. **Android Studio** includes a code editor for Kotlin, as well as other programming languages.

Code refactoring

**Code refactoring** is the process of restructuring existing computer code—changing the factoring—without changing its external behavior. This can make your code more readable and reduce its complexity so that it's easier to maintain.

Code snippet

A **code snippet** is a small chunk of reusable **source code**.

Coding conventions

**Coding conventions** are style guidelines and best practices for writing code. Coding conventions include how to name things, code formatting, and best practices for comments. Following the conventions for Kotlin makes your code easier to read, use, and maintain by other developers.

Conditional

**Conditional** statements determine the control flow of a program based on the **Boolean** outcome of an evaluation. Conditionals decide what code to run when the condition is true, and when the condition is false. Examples of conditional statements in Kotlin are if, else, and when.

Console

The **console**, in software development tools, is the text output for system messages. For example, in the [Kotlin Playground](https://play.kotlinlang.org/" \t "_blank), print statements output to the console.

Constraint

A **constraint** is a limitation or restriction. In the context of designing a layout using a ConstraintLayout for an **Android** app screen, a constraint represents a connection or alignment of a View to another View or the parent layout. For example, a View can be constrained to the top of the screen, or a button can be constrained between two other buttons in a row.

ConstraintLayout

A **ConstraintLayout** is a flexiblen Android ViewGroup that lets you position and size views using **constraints**.

Constructor

When you create an **instance** of a class, a **method** called a **constructor** is executed automatically. A constructor is responsible for doing everything necessary to make the **object** usable by the rest of the app. For example, a constructor will usually initialize (put the first value into) each field of the object which is being constructed. The only way to create an object is by calling (executing) a constructor for the object.

The name of a constructor is the same as the name of the class to which it belongs. A class can have more than one constructor, provided that each one has a different list of **parameters**.

Control Flow

By default, a computer executes code in a program from top to bottom, in the order in which the instructions are written. Some instructions, however, can direct execution to skip over other instructions, or to repeat other instructions; for example, if and repeat instructions. These instructions that cause program execution to deviate from a straight-line path are called the **control structure** of the code. The resulting path, or set of possible paths, is called the **control flow**.

Comments

**Comments** provide an explanation of source code that's meaningful only to the human reader. Comments must be surrounded by comment delimiters to identify them: that is, punctuation marks that tell the system not to read the enclosed text.

In Kotlin, comments that begin with a double slash // are called **inline comments**. An inline comment can be only one line long. Comments on subsequent lines must also begin with a double slash.

Comments enclosed by /\* and \*/ are called block comments, and can be many lines long.

Compile

Compilers **compile**, or translate, **source code** into instructions in the language that computers and mobile devices can understand and execute.

Compiler

A **compiler** is a special program that translates source code into instructions that computers and mobile devices can understand and execute.

Data type

A **data type** describes what kind of data the data is, and how it can be used. Integers (Int), **Booleans** (Boolean), strings (String), arrays (Array), and ranges (IntRange) are examples of data types. The data type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored.

Debugger

A **debugger** is a tool for **debugging** a software application or piece of code. It typically enables you to step through and examine values as the application executes.

Debugging

**Debugging** is the process of identifying and fixing errors in an app when it is not working as expected.

DP (Density-independent pixels)

The screen of an **Android device** is made of rows and columns of glowing dots called pixels. The density of a screen is a measure of how many pixels (or dots) there are per inch. Devices can range in screen density. For example, an mdpi (or medium density device) has 160 dots per inch, while an xxhdpi (extra extra high density device) has 480 dots per inch.

If you specify the size of views in pixel values to fit on the medium density device, then the same views would appear very small on the extra high density device. To save you from having to define a different layout for every density, you can express the size of your UI elements in a measure called density-independent pixel (dp or dip, pronounced "dee pee" or "dip"). 1 dp is equal to 1 pixel on an mdpi device. 1 dp is equal to 3 pixels on an xxhdpi device. Android devices will automatically convert from dp to the appropriate pixel values so that your layout looks right on the device.

Drawable

In an Android Studio app, a **drawable** is a **resource** that contains instructions for drawing something. This is usually a file that contains an image of various formats (Bitmap, PNG, JPG, SVG.). For example, to include a PNG image in your app, add the PNG as a drawable resource in your app. You can use the Resource Manager to add it. See [Drawable resources](https://developer.android.com/guide/topics/resources/drawable-resource" \t "_blank).

Empty string

An **empty string** is a **string** of zero length, represented as "". It is a character sequence of zero characters.

Encapsulation

**Encapsulation** means enclosing functionality that is logically related into a single place. One example of encapsulation is a **class**.

Function

A **function** is a discrete block of code that performs an operation and can return a value. In Kotlin, functions are declared using the fun keyword and can take **arguments** with either specified or default values. A function associated with a particular **class** is called a **method**.

Getters and Setters

**Functions** that retrieve (get) and set the values of properties, usually in **classes**. In Kotlin, **getters and setters** are provided for all class properties, saving you the work of having to write them yourself.

Hardcoded

**Hardcoded** values are values that are directly written into a program or app. It is a good coding practice to not use hardcoded values directly in code. For example, instead of putting strings into your Android app source code, you can store the strings in a separate file that collects all the strings for the app, making it much easier to change and translate them.

Import

A directive for including an **API** or other code that is not part of the current program into your code. Use import statements at the top of your Kotlin files to be able to use classes and APIs that were defined outside your app. For example, add import android.widget.TextView to use the TextView **class** in your code.

Keyword

A **keyword** is a special word used by Kotlin for its own purposes, and you cannot use a keyword as a **variable** name. For example, the word fun declares a **function**, and so you cannot use it as a variable name.

Layout

In Android Studio, The **Layout** is the arrangement of Views that make up the screen of an Android app.

IDE (Integrated Development Environment)

An **IDE**, or **Integrated Development Environment**, is a collection of tools for building software. **Android Studio** is an example of an **IDE**. The tools in Android Studio help you write your code and lay out how your app will look on the target device screen. There are tools to help you translate your app to other languages, tools to help you make it run faster, and tools to test your app on multiple virtual devices.

ImageView

An **ImageView** is a type of View that displays an image such as an icon or a photograph.

Instance (object instance)

An **instance** or **object instance** is an object created from a **class** definition. While the class acts as a blueprint, the instance is an actual thing that you can interact with in your code, such as accessing its properties or calling its **methods**.

Int

In Kotlin, **Int** is a **data type** for an integer number. In modern computers, integers are numbers between -2,147,483,648 and 2,147,483,647 (4 bytes or 32 bits per number).

Kotlin

**Kotlin** is a modern programming language with features that make it easier to be productive in writing concise code, which is also less prone to errors.

Line break

A **line break** creates a new line in text. In programming languages, such as Kotlin, the newline character is represented as '\n' and produces a line break when inserted in text strings.

main() function

The main() function of a Kotlin program is the entry point for program execution. When you run a Kotlin program, it always starts with the main() function. In an Android app, there is no explicit main() function for you to program beause your app code is executed by the Android system.

Method

A **method**is a function defined in a **class** and associated with the behavior of an **object instance** of the class.

Mockup

In the context of an app **user interface**, a **mockup** is an early design of the screens that does not show the final look or functionality.

Object

In**object-oriented programming**, an object refers to a particular **instance** of a **class**, where the object can be a combination of **variables**, **functions**, and data structures.

Object Oriented Programming (OOP)

**Object-oriented programming (OOP)** is a programming paradigm that focuses on **objects** rather than a specific set of **functions** or logic flow. This approach makes it easier to maintain large, complex applications. Dividing application data and code into discrete objects also makes it easier for teams to work on projects collaboratively.

Operators and operands

Math operators include plus (+), minus (-), and multiplication (\*). For an expression such a 5 + 3 = 8, the + is the operator while 2 and 3 are the operands. There are other types of operators. Logic operators include greater-than (>), equal-to (==), and not (!), and when applied, yield true or false, such as 5 > 3 being true.

Package

In program source code, **packages** are used to group related program elements such as **classes**, **variables**, and **functions**. In Kotlin, these elements are declared in files with a package declaration at the top of the file. To use elements that are part of a package in another package, you **import** the package.

The package name customarily contains only lowercase letters (no underscores) and separating dots and has to be globally unique. For example: package com.example.diceroller.

Parameter vs. argument

In the context of a **function** (or **method**), a **parameter** is variable declaration in the function definition. When the function is called, the **arguments** are the data you pass into its parameters.

Project

**Android Studio** organizes code into **projects** that contain everything that defines your **Android** app. This includes your app source code, build configurations, the manifest, **resources**, and code with tests.

Pseudocode

**Pseudocode** is well, not quite code. Its code-like language and structure is used to describe what an app might do, in a way that is easier to understand by humans. It can be useful for figuring out the correct approach to take for organizing code before all the details have been decided.

Reference

When you assign a number to a variable, the value is stored in the variable. When you assign an **object** to a variable, or pass it to a **function**, a **reference** to that object is used instead. The reference can be a unique ID, or the address of a location in memory. Using references instead of the copies of values and objects themselves can keep your code smaller.

Resources

In Android apps, **Resources** are additional files and content that your **code** uses, such as images, layout definitions, and user interface strings.

SP (Scalable pixel)

A **scalable pixel** (**sp**) is a unit for specifying the size of a font. The Android system calculates the actual font size to use based on the device and the user's preference set in the Settings app of their Android device. Always specify font sizes in sp units or scalable pixels.

String

Any series of characters—letters, numbers, punctuation marks, and spaces—is called a **string** of characters. The number of characters in the string is called the length of the string. A string may consist of words or of randomly chosen characters. It can also be a single character, or even no characters at all. The latter is called the **empty string**, and is the only string of length zero.

String Resource

A **resource** that is a **string**. In an Android app, strings that are displayed to the user should be defined in a strings.xml file.

String template

In a **string**, instead of printing text, you can include an embedded expression, that is, a directive to include the value of a variable or the result of a calculation using the ${variable} notation. This is called a **string template**.

System image

When you create an **Android Virtual Device (AVD)** emulator, you choose a system image to load for the device. The system image includes the Android operating system and **APIs (Application Programming Interface)** that enable specific features for the device.

TextView

A **TextView** is a type of View that displays text.

User Interface or UI

The **user interface** or **UI** of an **app** is what you see and interact with on the screen of an Android device. The user interface includes Views, buttons, and other interactive elements.

Val & var

Kotlin supports two types of **variables**: changeable and unchangeable. With **val**, you can assign a value once. If you try to assign something again, you get an error. With **var**, you can assign a value, then change the value later in the program.

Variable

A **variable** is a container that holds a value, such as a number or a piece of text. For example, a variable might hold the current score in a game or the name of a restaurant. Each variable has a name such as "currentScore" or "restaurantName". The contents of a variable can be changed as the app runs. That's why they're called "variables."

View

A **View** is a rectangular area on the screen. It has a width and height. There are many different types of Views. For example: An ImageView displays an image. A TextView displays text. A Button is a TextView that can be tapped. A **ViewGroup** is a View—often invisible to the user—that contains and positions other views inside of it, creating a **view hierarchy**.

ViewGroup

A **ViewGroup** is a **View** that can contain other views (called its children.) Any layout for an Android App always starts with a ViewGroup as the parent and contains child views.

View hierarchy

A **hierarchy** of views in an Android app layout is organized as a family tree of parents and children. The View at the top of the hierarchy, or root View, must be a ViewGroup container, such as a ConstraintLayout. The child views can be any views or view groups. You can layer the hierarchy as deep as necessary, but a flat view hierarchy with few levels of children is a best practice.

XML

**XML** stands for "Extensible Markup Language". It is a notation for writing information structured as a **hierarchy**. XML can be used to describe the structure of documents, vector graphics, and the hierarchy of **views** that make up the layout of an Android screen.

[**Back**](https://developer.android.com/codelabs/basic-android-kotlin-training-vocab/)

Interview questions

There are four Java classes related to the use of sensors on the Android platform. List them and explain the purpose of each

The four Java classes related to the use of sensors on the Android platform areL

* Sensor: Provides methods to identify which capabilities are available for a specific sensor.
* SensorManager: Provides methods for registering sensor event listeners and calibrating sensors.
* SensorEvent: Provides raw sensor data, including information regarding accuracy.
* SensorEventListener: Interface that defines callback methods that will receive sensor event notifications.
* A ContentProvider manages access to a structured set of data. It encapsulates the data and provide mechanisms for defining data security. ContentProvider is the standard interface that connects data in one process with code running in another process.

onPause() and onStop() will not be invoked if finish() is called from within the onCreate() method. This might occur, for example, if you detect an error during onCreate() and call finish() as a result. In such a case, though, any cleanup you expected to be done in onPause() and onStop() will not be executed.

Although onDestroy() is the last callback in the lifecycle of an activity, it is worth mentioning that this callback may not always be called and should not be relied upon to destroy resources. It is better have the resources created in onStart() and onResume(), and have them destroyed in onStop() and onPause(), respectively.

Common use cases for using an Intent include:

* **To start an activity**: You can start a new instance of an Activity by passing an Intent to startActivity() method.
* **To start a service**: You can start a service to perform a one-time operation (such as download a file) by passing an Intent to startService().

DDMS is the [Dalvik Debug Monitor Server](http://developer.android.com/tools/debugging/ddms.html" \t "_blank) that ships with Android. It provides a wide array of debugging features including:

* port-forwarding services
* screen capture
* thread and heap information
* network traffic tracking
* incoming call and SMS spoofing
* simulating network state, speed, and latency
* location data spoofing
* **To deliver a broadcast**: You can deliver a broadcast to other apps by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

The Intent object is a common mechanism for starting new activity and transferring data from one activity to another. However, you cannot start a ContentProvider using an Intent.

When you want to access data in a ContentProvider, you must instead use the ContentResolver object in your application’s Context to communicate with the provider as a client. The ContentResolver object communicates with the provider object, an instance of a class that implements ContentProvider. The provider object receives data requests from clients, performs the requested action, and returns the results.

An [activity](http://developer.android.com/reference/android/app/Activity.html) is typically a single, focused operation that a user can perform (such as dial a number, take a picture, send an email, view a map, etc.). Yet at the same time, there is nothing that precludes a developer from creating an activity that is arbitrarily complex.

Activity implementations can optionally make use of [the Fragment class](http://developer.android.com/reference/android/app/Fragment.html) for purposes such as producing more modular code, building more sophisticated user interfaces for larger screens, helping scale applications between small and large screens, and so on. Multiple fragments can be combined within a single activity and, conversely, the same fragment can often be reused across multiple activities. This structure is largely intended to foster code reuse and facilitate economies of scale.

A fragment is essentially a modular section of an activity, with its own lifecycle and input events, and which can be added or removed at will. It is important to remember, though, that a fragment’s lifecycle is directly affected by its host activity’s lifecycle; i.e.,

Serializable is a standard Java interface. You simply mark a class Serializable by implementing the interface, and Java will automatically serialize it in certain situations.

Parcelable is an Android specific interface where you implement the serialization yourself. It was created to be far more efficient than Serializable, and to get around some problems with the default Java serialization scheme.