Vocabulary for Android Basics in Kotlin

This vocabulary list explains the terminology used in the Android Basics in Kotlin course (https://developer.android.com/courses/android-basics-kotlin/course).

Learn more

- See the Kotlin Quick Guide for Android Basics in Kotlin (https://developer.android.com/courses/android-basics-kotlin/kotlin-quick-guide/) for concepts in this course in code.
- See the Kotlin Language Documentation (https://kotlinlang.org/docs/reference/) for full reference.

A

Abstract class

An **abstract class** is a class that is marked with the `abstract` keyword in Kotlin. An abstract class may have properties or methods that have not been implemented yet. In that case, initializing the abstract property or implementing the abstract method is up to the subclasses. Because of this, you cannot instantiate an abstract class directly

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.

Adaptive icons

Adaptive icons is an app icon format for Android apps which can display a variety of shapes across different device models. It was introduced in the Android O release, and is intended to make all the app icons have a uniform shape on a device.

adb (Android Debug Bridge)

adb, or **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

An **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**.

Annotations

Annotations can add additional information to classes, methods, or parameters to provide hints about the data types. For example, the `@StringRes` annotation tells Android Studio that a variable holds a resource ID for a string, and the `@DrawableRes` annotation indicates that this variable holds a resource ID for a drawable.

Any

All classes in Kotlin derive from the **superclass** `Any` so every instance of a class also has a type of `Any`. The `Any` class contains implementations of the methods `equals()`, `hashCode()`, and `toString()`, so these methods can be called on any class instance or overridden in class definitions. See superclass.

APK (Android Application Package)

APK, or **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 Manager** is an interface you can launch from **Android Studio** that helps you create and manage **Android Virtual Devices (AVDs)**.

B

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.

C

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`.

Chaining

Chaining is combining two or more method calls or property accesses into a single expression. This helps reduce the use of temporary variables and can make your code easier to read and maintain. For example, instead of assigning a number property to a variable, then converting the variable to a string, it can be done in a single expression.

```
// Separate method calls
Double cost = binding.costOfService
String costString = cost.toString()

// Chained method calls
String costString = binding.costOfService.toString()
```

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 Handler

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

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**.

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.

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.

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://developer.android.com/training/kotlinplayground), 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 flexible 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**.

D

Dark theme

Dark theme is an alternate UI mode for Android which uses light text on dark backgrounds instead of dark text on a light background as in a light theme. It can reduce power usage (depending on the device's screen technology) and can make it easier to use a device in a low-light environment.

Data class

A **data class** is a specific type of Kotlin class that is intended to hold data. For classes marked with the `data` keyword, the Kotlin compiler automatically generates some helpful utility functions such as `equals()`, `hashCode()`, `toString()`, and `copy()` methods based on the properties in the primary constructor.

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.

Double

Double is a basic data type to represent decimal numbers in Kotlin, as opposed to an `Int`, which can only represent integer values.

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).

E

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**.

Exception

An **exception** is the system's way of saying there is a problem. It often occurs when the data a function or class receives is different from what is expected. For example, the `String.toDouble()` function will generate a `NumberFormatException` if the string is not a valid representation of a number.

F

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**.

G

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.

Gradle

Gradle is a tool used by Android Studio to automate and manage the build process for your app. Gradle takes the **source code** you write for an app, along with platform code, **libraries**, and **resources**, and assembles all these parts into an **APK** that can be installed and run on an Android device. Modify the `build.gradle` files in your project to configure aspects of your project such as the platform versions your app can run on and settings for how your app should be built.

H

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.

I

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.

Immutable

An **immutable** object can't be changed after it is created. See also **mutable**.

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.

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).

K

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.

L

lateinit

Normally non-null properties must be initialized in the constructor, but sometimes that isn't practical. In those cases, a property can be marked with **`lateinit`**, which is a promise that the variable will be initialized before it is used. Note that an exception will be thrown if that property is accessed before being initialized.

Layout

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

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**.

List

A **list** is an ordered set of elements that can be accessed by their numeric index in the list. In Kotlin, a list can be either mutable or immutable depending on how it was defined. Mutable lists can be changed both in terms of the number of elements, and the elements themselves; immutable lists cannot.

M

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 because your app code is executed by the Android system.

Material

Material (https://material.io/design/introduction) is a design system inspired by the physical world and its textures, including how objects reflect light and cast shadows. It provides design guidelines on how to build your app UI in a readable, attractive, and consistent manner to create an overall compelling user experience.

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.

Mutable

A **mutable** object can be changed after it is created. See also **immutable**.

N

Null

Null is a special value that means "no value". It's different from a default value, for example, a `Double` having a value of `0.0` or an empty `String` with zero characters, "". `null` means there is no value. If a method expects a value, passing `null` will cause an error. In Kotlin, it's preferable to not allow `null` values and you have to use special operators to permit a `null` value. See also **exception**.

O

Object

In **object-oriented programming**, **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`.

P

Package

In **program** source code, **packages** are used to group related program elements such as **classes** and **functions**. A package provides a **namespace**, a way for the names of classes to only need to be unique in that package and not globally. In Kotlin, you'll find a package declaration at the top of files indicating the package name. To use elements that are part of a different package, you need to **import** the other package. See also **qualified name**.

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.

Private

By default, members of a class are **`public`**, meaning other code can access them. However, it's considered a best practice to limit outside access to just the essentials. Marking a method or property **`private`** means it can only be accessed from inside that class. This helps to avoid

exposing parts of the implementation that you didn't intend for other callers to have access to or depend on.

Program

A **program** is a series of instructions that a computer system executes to accomplish some action.

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 not quite "real" 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.

Public

By default, members of a class are **public**, meaning other code can access them. See also **private**.

Q

Qualified name

The **qualified name** of a class consists of the full package name and the class name. For example, the fully qualified name of `TextView` would be `android.widget.TextView`. The qualified name is generally only used if there is a conflict between a class or function in this namespace and an imported namespace. See also namespace.

R

RecyclerView

A `RecyclerView` is a widget in Android you can use to display a list of items. Implement a custom `RecyclerView.Adapter` to provide the `RecyclerView` with the appropriate `ViewHolder` for each item in the data source. Instead of creating a new view for every item in the list, the `RecyclerView` recycles views as needed.

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.

S

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.

Stack trace

A **stack trace** is a list of which methods were called before your program pauses or stops execution. If it stopped because of an exception, the stack trace can show which method the exception occurred in.

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**.

Subclass / inheritance / superclass

A **class** is like a blueprint for an **object**, but it can also be a blueprint for creating other classes with similar functionality or a different focus. You are creating a **subclass** (child class) when you define a new class using another class as a template. The subclass **inherits** properties and functions of its **superclass** (parent class). It can also provide its own implementation by overriding properties and functions that exist in the superclass. One common example is the `toString()` method. It is first defined in the `Any` class, but it is common to provide your own implementation of the `toString()` method in subclasses. A subclass also has the ability to call public functions and properties of the superclass.

System image

When you create an **Android Virtual Device (AVD)** to simulate in the 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.

T

TextView

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

U

User Interface (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.

V

val and 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."

Vector drawable

A **vector drawable** is a vector graphic defined in an XML file as a set of points, lines, and curves along with its associated color information. Instead of storing the information to draw an image at a specific size like a bitmap image, a vector graphic stores the instructions to draw an image at any size. A vector drawable can be scaled without quality loss and is often smaller

than a comparable bitmap image. That means the same file is resized for different screen densities without loss of image quality or needing additional files.

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.

View binding

View binding is a feature provided by Android Jetpack that lets you more easily write code to interact with views. Enable view binding in your project's app module to generate a binding class for each XML layout file. An instance of a binding class contains direct references to all views that have a resource ID in the corresponding layout. This can be used instead of calling `findViewById()` in most cases.

ViewGroup

A **ViewGroup** is a `View`—often invisible to the user—that contains and positions other views inside of it (called its children), creating a **view hierarchy**. 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.

X

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.

Last updated 2020-09-30 UTC.