Terms	Definition
Action	The intent also incorporates a URI (a Uri object) that references the data to be acted upon, and potentially, the MIME type of that data. The data's type is often determined by the intent's action; for example, when the action is "ACTION_EDIT," the data should contain the URI of the document to edit.
Activate components	An intent can initiate activities, services, and broadcast receivers in two ways: by explicitly specifying the target component or by using an implicit intent to define the action type and optional data. In either case, the system identifies an appropriate component and initiates it, giving the user the option to select the one to fulfill the intent.
Activities	An Activity represents a single screen with a user interface, like the various screens in an email app for composing, reading, and managing emails. Each activity operates independently and can be launched by different apps when permitted. Activities play a crucial role in mediating interactions between the system and the app. They keep track of user interactions, prioritize processes that have been paused, assist apps in managing process termination, and facilitate seamless user navigation between apps and the Android system. Implementing an activity typically involves creating a subclass of the Activity class.
anim/	XML files that define Tween animations.
Animate between activities	On Android 5.0 (API level 21) and higher, you also have the capability to design animations that transition between your activities. This functionality is based on the same transition framework outlined in the preceding section, enabling you to create animations that switch between layouts in separate activities.
Animate layout changes	Starting from Android 4.4 (API level 19) and onward, you can employ the transition framework to create animations while transitioning between layouts within the current activity or fragment. You simply specify the starting and ending layouts and define the desired animation type. The system then takes care of executing an animation between the two layouts. This feature allows you to either replace the entire user interface or modify specific views by moving or replacing them.
Animate UI visibility and motion	When you need to modify the visibility or position of views within your layout, incorporating subtle animations helps users comprehend how the user interface is evolving.
animator/	XML files that define Property animations.
App components	Android applications can be developed using the Kotlin, Java, and C++ programming languages. The Android SDK tools are responsible for compiling code into either APKs or AABs. APKs encompass all the necessary app content required for runtime installation on Android devices, while AABs contain additional metadata. Google Play servers generate optimized APKs tailored to specific device requirements,

	ensuring the correct installation of the app.
App resources	Android app requires distinct resources such as images, audio files, and visual presentation elements. These resources enable easy updates and optimization for various device configurations, such as different languages and screen sizes, all without the need to modify the source code.
App resources overview	Resources in Android encompass supplementary files and static content employed within your code. These resources can include bitmaps, layout definitions, user interface strings, and more. It's good practice to externalize your app's resources, like images and strings, to promote code independence and offer alternative resources tailored for specific device configurations. Resources are accessed using resource IDs generated within your project's R class, and they can be organized to cater to particular device configurations.
Application fundamentals	Android applications can be developed using the Kotlin, Java, and C++ programming languages. The Android SDK tools are responsible for compiling code into either APKs or AABs. APKs encompass all the necessary app content required for runtime installation on Android devices, while AABs contain additional metadata. Google Play servers generate optimized APKs tailored to specific device requirements, ensuring the correct installation of the app.
App-specific storage	You can save files exclusively intended for your app within dedicated directories in either the internal storage volume or specialized directories in external storage. The internal storage directories are particularly secure for safeguarding sensitive information, preventing unauthorized access by other applications.
Attributes	The XML vocabulary in Android enables the rapid design of UI layouts and screen elements, much like creating web pages with HTML. Each XML file must include a root element, to which child elements can be gradually added.
Broadcast receivers	A Broadcast Receiver is a component that enables the system to deliver events to an app outside the regular user interaction flow. This allows the app to respond to system-wide broadcast announcements. Broadcast receivers can be system-generated or initiated by apps themselves. They serve various purposes, from creating status bar notifications to acting as gateways for other components, such as scheduling a JobService based on an event. Since broadcast receivers often involve interactions with the app, it's vital to consider security implications when configuring them. They are implemented as subclasses of BroadcastReceiver, and each broadcast is delivered as an Intent object.
Building an intent	An Intent object contains information crucial for the Android system to determine which component to launch. This includes details such as the exact component name or component category that should receive the intent. Additionally, it carries information necessary for the recipient component to execute the intended action effectively, such as the action to perform and the data to act upon.

Categories of storage locations	Android offers two primary physical storage locations: internal storage and external storage. While internal storage is typically smaller in capacity compared to external storage on most devices, it stands out for its universal availability, making it a more dependable choice for storing data critical to your app's functionality.
Category	Intent descriptions contain a string indicating the component responsible for handling the intent, with common categories such as "Category_BROWSABLE" for activities initiated by web browsers and "Category_LAUNCHER" for the initial task activity listed in the system's application launcher.
color/	XML files that define a state list of colors.
Common layouts	A very subclass of the ViewGroup class offers a unique way to display the nested views within it. The ConstraintLayout is the most versatile layout type and offers the best tools for maintaining a shallow layout hierarchy.
Component name	Explicit intent delivery to the app component hinges on the presence of the component name; in its absence, the intent becomes implicit and is determined by other intent information. A string in the intent specifies the generic action to be executed, like "view" or "pick."
Configuring the manifest	To enable your app to utilize activities, you must declare these activities and certain attributes within your app's manifest.
Content providers	A content provider is a system-designed entity that manages a shared set of app data, such as user contact information, which can be stored in various storage locations. It allows other apps to query or modify the data if permitted. Content providers are implemented as subclasses of ContentProvider and must implement a standard set of APIs for other apps to perform transactions.
Data and file storage overview	Android employs a file system structure reminiscent of disk-based systems found on various platforms. Within this system, multiple options are available for preserving your app's data.
Databases	For managing structured data, the recommended approach is to use the Room persistence library to create a private database.
Declare activities	To declare an activity, navigate to your manifest file and include an <activity> element as a child of the <application> element.
Declare app requirements	An Intent is an asynchronous message that triggers activities, services, and broadcast receivers within an app. It is created using an Intent object, which defines a message to activate either an explicit or implicit intent.
Declare component capabilities	To guarantee that your app is installed on devices it's compatible with, you should define a profile that outlines the supported device types. This is accomplished by specifying device and software requirements within your app's manifest file. External services, such as Google Play,

	review these declarations to facilitate effective filtering.
Declare intent filters	Intent filters are a potent feature of Android, allowing the system to launch activities in response to explicit or implicit requests. They come into play when the system UI prompts the user to choose which app should handle a particular task.
Declare permissions	The <activity> tag in your manifest can be used to control which apps can initiate a specific activity. A parent activity cannot start a child activity unless both activities possess the same permissions defined in their respective manifests. If you declare a <uses-permission> element for a parent activity, each child activity should also have a matching <uses-permission> element.
Declare UI elements in XML.	Android offers a straightforward XML vocabulary that closely aligns with the View classes and subclasses, including widgets and layouts. You can also utilize Android Studio's Layout Editor, which provides a drag-and-drop interface for creating XML layouts.
Define the location update callback	The fused location provider triggers the LocationCallback.onLocationResult() callback method, passing a list of Location objects containing latitude and longitude data. Below, you'll find an example of how to implement the LocationCallback interface, define the method, and retrieve the location update's timestamp, as well as display the latitude, longitude, and timestamp on your app's user interface.
Delivering a broadcast	A broadcast is a message accessible to any app within the system. The system generates broadcasts to signal significant system events, such as system boot-up or device charging initiation. To transmit a broadcast to other apps, you can achieve this by forwarding an Intent using either sendBroadcast() or sendOrderedBroadcast().
Drawable	To establish an alias for an existing drawable, employ the <drawable> element.
drawable/	Bitmap files (PNG, .9.png, JPG, or GIF) or XML files that are compiled into the following drawable resource subtypes: Bitmap files, Nine-patches (re-sizable bitmaps), State lists, Shapes, and Animation drawables.
Explicit intents	To specify which application should handle the intent, you can supply either the target app's package name or a fully-qualified component class name. Explicit intents are typically employed when initiating a component within your own app, as you have knowledge of the specific activity or service you wish to start. For instance, you may trigger a new activity in response to a user's action or initiate a background service for file downloads.
Extend and customize a style	When crafting your own styles, it's advisable to build upon existing styles from the framework or Support Library. Doing so ensures compatibility with the platform's UI styles. To extend a style, specify the parent style using the "parent" attribute. This allows you to override

	inherited style attributes while introducing new ones to achieve your desired visual effects.
Extras	Extras are key-value pairs that provide additional information for specific actions. They can be added using putExtra() methods or created as a Bundle object. For instance, an intent to send an email can specify the recipient and subject. To declare extra keys, include the app's package name as a prefix.
Flags	A n explicit intent is employed to initiate a precise app component, like a specific activity or service within your app. Crafting an explicit intent involves defining the component name for the Intent object, with all other intent properties being optional.
font/	Font files with extensions such as TTF, OTF, or TTC, or XML files that include a <font-family> element.
Get the last known location	Initiating periodic location updates is best preceded by obtaining the last known location of the device, ensuring your app has an initial point of reference. You can achieve this by calling getLastLocation(), as demonstrated in the "Getting the Last Known Location" lesson. The subsequent sections assume that your app has successfully retrieved the last known location and stored it as a Location object within the global variable mCurrentLocation.
ID	Any View object can be associated with an integer ID, making it uniquely identifiable within the tree. During compilation, this ID is referenced as an integer, but it's typically assigned in the layout XML file as a string in the "id" attribute, a common XML attribute shared by all View objects.
Implicit intents	Conversely, when you don't need to specify a particular component, you can declare a general action to perform, allowing a component from another app to handle it. For example, if your objective is to display a location on a map to the user, you can use an implicit intent to request that another capable app show the specified location on a map.
Instantiate layout elements at runtime	Your app can dynamically create View and ViewGroup objects and manipulate their properties programmatically.
Intents and intent filters	An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases:
Introduction to activities	The Activity class plays a pivotal role in Android app development, and understanding how activities are launched and orchestrated is essential to grasp the Android application model. Unlike conventional programming paradigms where apps typically commence with a main() method, Android initiates code execution within an Activity instance by invoking specific callback methods that correspond to distinct stages of its lifecycle.

Introduction to animations	Animations serve as visual cues that inform users about ongoing events in your app. They prove particularly valuable when the user interface undergoes changes in state, like the arrival of new content or the availability of new actions. Additionally, animations enhance your app's overall appearance, contributing to a more polished and high-quality look and feel.
Layout	To craft an alias for an existing layout, employ the <include> element, encapsulated within a <merge>.
Layout position	A view is defined by its rectangular geometry, which includes a location (specified as left and top coordinates) and dimensions (specified as width and height) in pixels.
layout/	XML files that define a user interface layout.
Layouts in Views	A layout in Android defines the structure of a user interface within your app, typically within an activity. It's constructed using a hierarchy of View and ViewGroup objects. A View is typically responsible for rendering something that the user can see and interact with.
Make a location request	To commence location updates, your app must first establish a connection with location services and define its location request parameters. The "Changing Location Settings" lesson provides guidance on this process. Once a location request is set up, you can initiate regular updates by invoking requestLocationUpdates().
menu/	XML files that define app menus, such as an options menu, context menu, or submenu.
mipmap/	Drawable files for different launcher icon densities.
onCreate()	As onCreate() concludes, the activity enters the Started state and becomes visible to the user. This callback involves final preparations for the activity's foreground appearance and interaction.
onDestroy()	The system calls onDestroy() before an activity is completely destroyed to ensure that all resources are released. This is the final callback in the activity's lifecycle, essential for proper lifecycle management.
onPause()	An activity is halted by the system when it is no longer visible to the user, either due to destruction, a new start, or resumption. The next callback is either onRestart() or onDestroy() if the activity is completely terminated.
onRestart()	The onRestart() callback is invoked when an activity is about to restart, restoring its state from its stop, and it is always followed by onStart().
onResume()	When an activity loses focus and transitions to the Paused state, indicating that the user may be leaving the activity, the system calls onPause(). Although this state may still allow updates to the UI if the user expects it, onPause() should not be used for tasks like data saving, network calls, or database transactions. The next callback in the sequence is either onStop() or onResume().

onStart()	As onCreate() concludes, the activity enters the Started state and becomes visible to the user. This callback involves final preparations for the activity's foreground appearance and interaction.
onStop()	An activity is halted by the system when it is no longer visible to the user, either due to destruction, a new start, or resumption. The next callback is either onRestart() or onDestroy() if the activity is completely terminated.
Physics-based motion	Whenever feasible, apply real-world physics principles to your animations to make them appear natural. They should retain momentum when the target changes and offer seamless transitions during any alterations. Two common physics-based animations are spring animations and fling animations.
Preferences	For private, basic data, you can employ key-value pairs to manage and store the information.
raw/	Arbitrary files to save in their raw form.
Request location updates	Effective utilization of location information can greatly enhance the user experience with your app. Whether your app assists users in navigation, tracking assets, or any other location-based functionality, regular access to the device's location is often essential. Beyond latitude and longitude, your app may require additional data like bearing, altitude, or velocity. This wealth of information is readily accessible through the Location object, obtainable from the fused location provider.
Save the state of the activity	Alterations in the device's configuration, such as adjustments in screen orientation or language, may lead to the termination of the current activity. Consequently, your app should retain all essential information required for reestablishing the activity. One effective method for achieving this is by utilizing an instance state stored in a Bundle object.
Scoped storage	Starting from Android 10 (API level 29) and above, apps are designed with scoped access into external storage, known as scoped storage, as the default configuration. This approach is aimed at enhancing users' control over their files while reducing clutter. Apps with scoped storage have access restricted solely to their designated app-specific directory on external storage, along with specific types of media that the app has generated.
Services	A Service is a component designed to keep an app running in the background for various purposes, such as performing long-running tasks or servicing remote processes. Services operate without a user interface and can perform tasks like playing music or fetching data without disrupting user interactions. There are two main types of services: started services and bound services.
Shared storage	Store files that your app intends to share with other apps, including media, documents, and other files.

Size, padding, and margins	The size of a view is characterized by its width and height, each represented as pairs of values.
Starting a service	A Service is a behind-the-scenes component dedicated to executing tasks without a user interface. Starting from Android 5.0 (API level 21) and beyond, you can initiate a service using JobSchedulerYou can trigger a service to perform single operations, such as file downloads, by employing startService() and passing an Intent. This Intent specifies the service to activate and transports any requisite data.
Starting an activity	An Activity serves as a representation of an individual screen within an app. You can initiate a new instance of an Activity by passing an Intent to the startActivity() method. This Intent outlines the specific activity to launch and can transport any required data.
Stop location updates	It's essential to consider whether you should halt location updates when the activity loses focus, such as when the user switches to another app or a different activity within the same app. This can be beneficial for conserving power, particularly if your app doesn't require continuous location data in the background. This section illustrates how to stop location updates in the activity's onPause() method.
Strings and other simple values	For an alias to an existing string, employ the resource ID of the desired string as the value for the new string.
Styles	A style serves as a collection of attributes defining the appearance of a single View. These attributes range from font color and size to background color and beyond.
Themes	A theme encompasses a collection of attributes that's applied to the entire app, activity, or view hierarchy, not just to a single view. When you apply a theme, all views within the app or activity inherit and apply each of the theme's supported attributes. Furthermore, themes can extend their influence to styles for non-view elements like the status bar and window background.
values/	XML files that contain simple values, such as strings, integers, and colors.
View files on a device	To view the files stored on a device, use Android Studio's Device File Explorer
Write the XML	The XML vocabulary in Android enables the rapid design of UI layouts and screen elements, much like creating web pages with HTML. Each XML file must include a root element, to which child elements can be gradually added.
xml/	Arbitrary XML files that can be read at runtime by calling Resources.getXML().