



El futuro digital
es de todos

MinTIC

Unidad 4





El futuro digital
es de todos

MinTIC

Tema 2 - Paleta gráfica



Componentes gráficos de la paleta de Android Studio



Los componentes gráficos de la paleta de Android Studio se clasifican en 7 grupos:

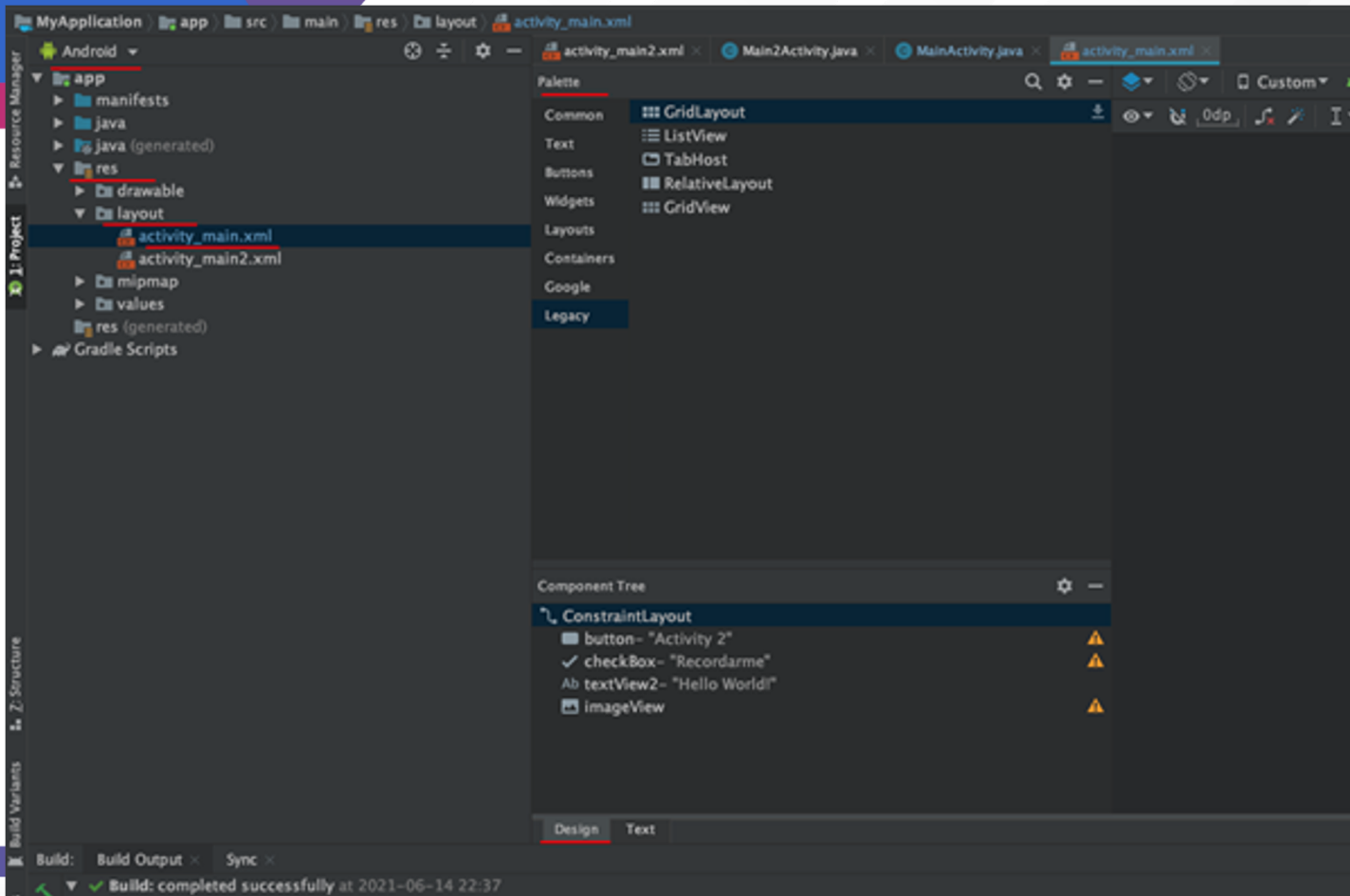
1. Common
2. Text
3. Buttons
4. Widgets
5. Layouts
6. Containers
7. Google
8. Legacy

La paleta de Android Studio contiene diferentes vistas que podemos arrastrar al "editor de diseño" que representa la pantalla de un dispositivo Android. Estas vistas se dividen en categorías.



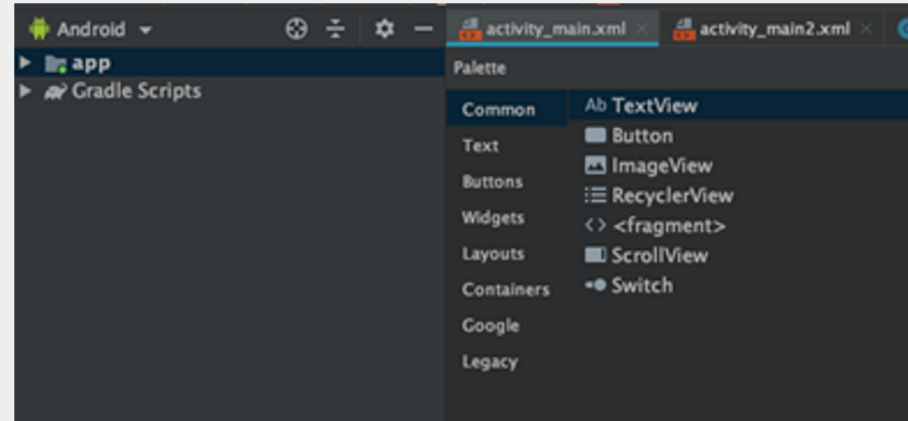


Luego en la parte inferior al abrir el archivo xml nos permitirá pasar a modo “Design”, luego podremos ver los siguientes componentes en diferentes grupos:



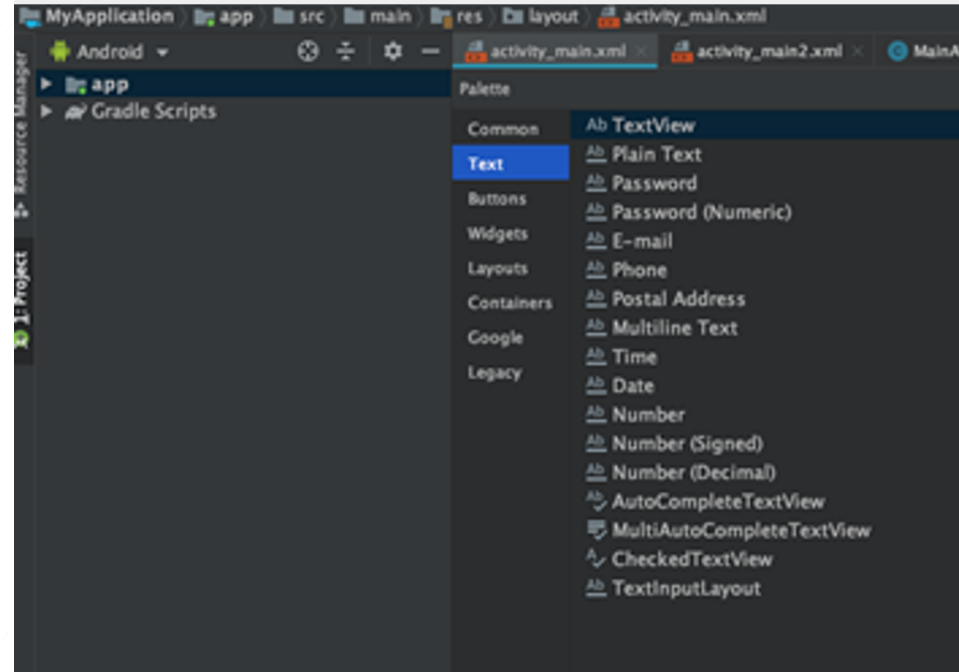
1 - Common

Este grupo contiene los componentes más comunes utilizados para el desarrollo de Apps entre ellos podemos encontrar con el TextView, Button, ImageView entre otros.



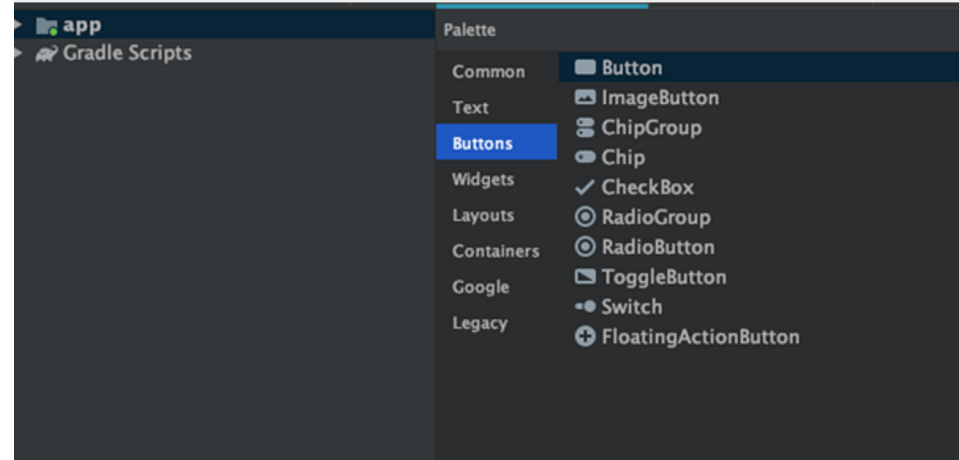
2 - Text

Este grupo de componentes nos proporciona diferentes tipos de TextView para usarlos en las diferentes opciones de una App.



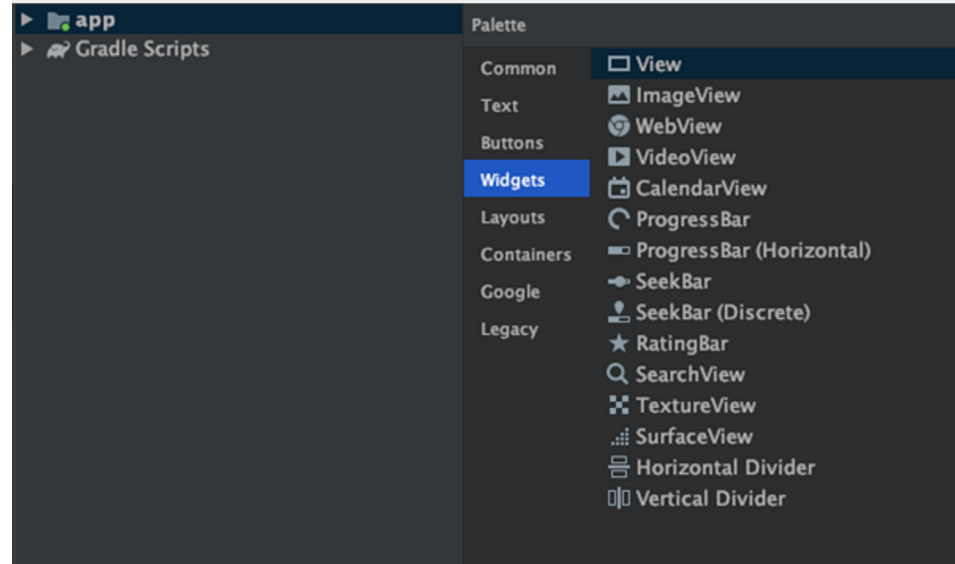
3 - Button

En este grupo de elementos se pueden encontrar todos los diferentes tipos de botones que pueden ser usados en una App



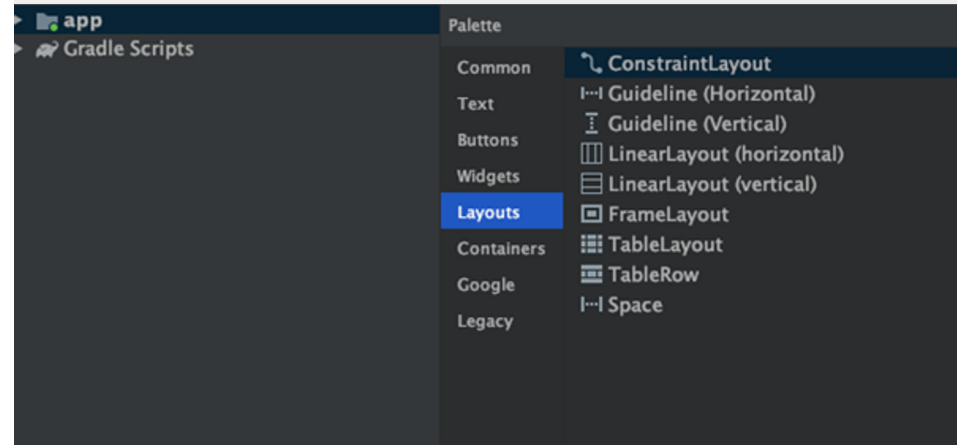
4 - Widgets

En este grupo de elementos podemos encontrar diferentes componentes para mejorar la experiencia de usuario y tener múltiples alternativas a la hora del desarrollo de una App



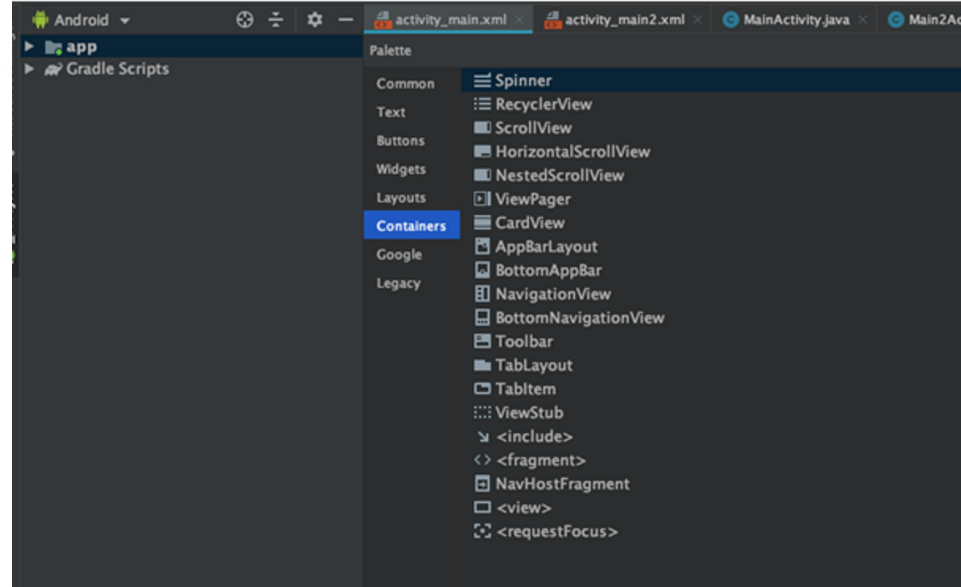
5 - Layouts

En este grupo de elementos podemos encontrar las diferentes opciones de componentes que tenemos para la construcción visual de una App en cuanto a la distribución de los elementos en la pantalla como son los ConstrainLayout, LinearLayout y los FrameLayout entre otros



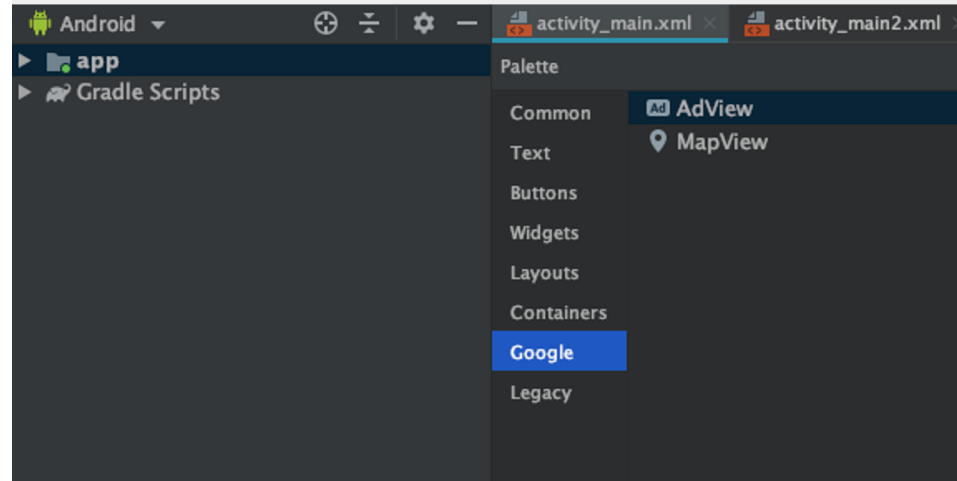
6 - Containers

En este grupo de elementos podemos encontrar las diferentes componentes que existen para cargar información en las diferentes pantallas de una App.



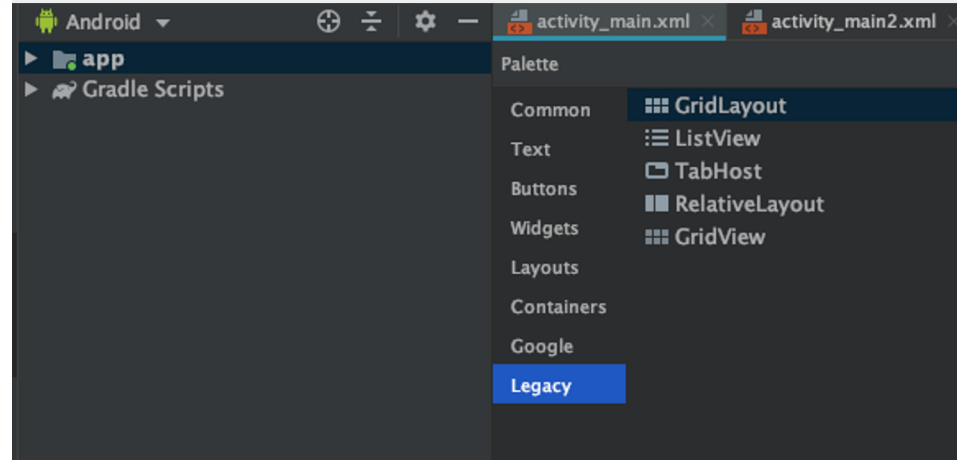
7 - Google

En este grupo de elementos podemos encontrar componentes que pueden ser usados con las Api de Google.



8 - Legacy

En este último grupo de elementos se encuentran aquellos que desde Android Studio ya se consideran que su uso está anticuado o están en desuso.





Cómo abrir documentación

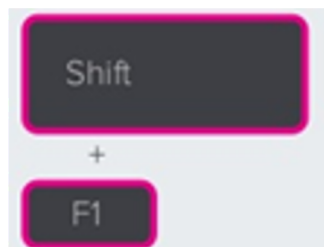
Si quieres abrir la documentación **de referencia** para desarrolladores de Android correspondiente a una vista o un grupo de vistas:

Selecciona el elemento de la IU en Palette y presiona **Shift + F1**.

Si quieres abrir la documentación de **lineamientos** sobre el material correspondiente a una vista o un grupo de vistas:

Haz clic con el botón derecho en el elemento de la IU ubicado en Palette y selecciona la opción Material Guidelines del menú contextual.

Si no existe una entrada específica para el elemento, este comando abrirá la página principal de la documentación de lineamientos sobre el material.





Cómo buscar elementos

Para buscar una vista o un grupo de vistas por nombre en **Palette**, haz clic en el botón **Search** , en la parte superior de **Palette**.

También puedes escribir el nombre del elemento cuando la ventana **Palette** está activa.

- Puedes ver los elementos de uso frecuente en la categoría **Common**, en **Palette**.
- Para agregar un elemento a esta categoría, haz clic con el botón derecho en una vista o un grupo de vistas en **Palette** y, luego, haz clic en la opción **Favorite** del menú contextual.



El futuro digital
es de todos

MinTIC

Tema 3 - Button



La clase Button

Un botón es un control con texto o imagen que realiza una acción cuando el usuario lo presiona

La clase Java que lo represente es Button y puedes referirte a él dentro de un layout con la etiqueta `<Button>` respectivamente



AGREGAR

Para agregar un botón a la actividad principal de tu app **dirígete a activity_main.xml** y agrega la siguiente definición.

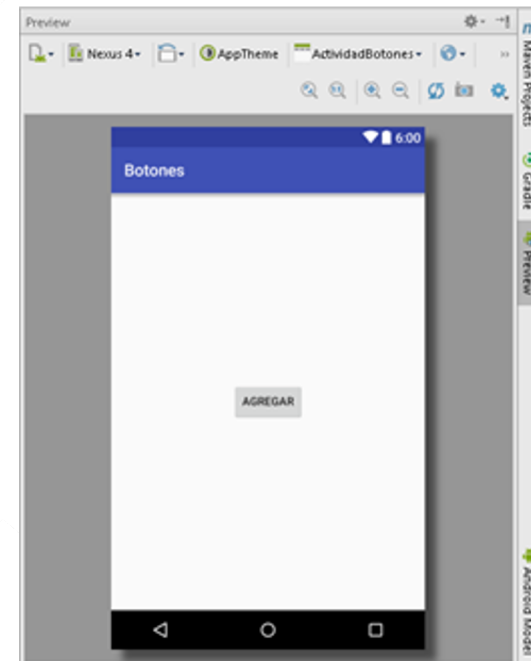
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="AGREGAR" />

</RelativeLayout>
```

El texto que especifica su acción
especificalo en el atributo
android:text. Para este caso:
AGREGAR

Con ello centramos el
botón en el **relative**
layout.





Atributos de un botón

Si quieres cambiar las propiedades de un botón recurre a los atributos que la documentación presenta en formato Java o XML.

Debido a que Button extiende de TextView, puedes usar todos los atributos de esta clase

Atributo	Descripción
<code>android:text</code>	Permite cambiar el texto de un botón
<code>android:background</code>	Se usa para cambiar el fondo del botón. Puedes usar un recurso del archivo <code>colors.xml</code> o un <code>drawable</code> .
<code>android:enabled</code>	Determinar si el botón está habilitado ante los eventos del usuario. Usa <code>true</code> (valor por defecto) para habilitarlo y <code>false</code> en caso contrario.
<code>android:gravity</code>	Asigna una posición al texto con respecto a los ejes x o y dependiendo de la orientación deseada. Por ejemplo: Si usas <code>top</code> , el texto se alineará hacia el borde superior.
<code>android:id</code>	Representa al identificador del botón para diferenciar su existencia de otros views.
<code>android:onClick</code>	Almacena la referencia de un método que se ejecutará al momento de presionar el botón.
<code>android:textColor</code>	Determina el color del texto en el botón
<code>android:drawable</code>	Determina un <code>drawable</code> que será dibujado en la orientación establecida. Por ejemplo: Si usas el atributo <code>android:drawableBottom</code> , el <code>drawable</code> será dibujado debajo del texto.



Cambiar Texto de un Boton

```
<Button
```

```
    android:id="@+id/button"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_centerHorizontal="true"
```

```
    android:layout_centerVertical="true"
```

```
    android:text="Agregar"
```

```
    android:textAllCaps="false"/>
```

Por defecto el texto del botón estará en mayúsculas, pero si quieres deshabilitar esta característica usa el valor false en el atributo android:textAllCaps.



Cambiar texto programáticamente

Paso 1

Abre **ActividadBotones.java**

Paso 2

Obtén la instancia del botón
con **findViewById()**

Paso 3

Y luego invoca **setText()** con
una secuencia de caracteres
como parámetro.



Cambiar texto programáticamente

```
public class ActividadBotones
    extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button boton = findViewById(R.id.button);
        boton.setText("Ordenar");
    }
}
```



ORDENAR



Cambiar texto programáticamente

En caso de XML usa la notación de recurso @string o @android:string (strings del sistema) de la siguiente forma:

```
<Button  
...  
    android:text="@string/texto_agregar"/>
```

En Java solo usa el operador punto para llegar al identificador perteneciente a la clase R:

```
Button boton = findViewById(R.id.button);  
boton.setText(R.string.texto_agregar);
```



Cambiar el color del fondo

Ejemplo

Usar el color primario del proyecto como color de background de un botón.

Solución

Invoca la referencia `@color/colorPrimary` de tu archivo `values/colors.xml`:

```
<Button  
    ...  
    android:background="@color/colorPrimary"/>
```

Resultado



AGREGAR



Cambiar el color del fondo

Sin embargo, hacer esto hace perder la reacción de superficie que se tenía antes por el Material Design.

La forma redondeada tampoco se hace presente.

Solo queda la elevación al momento de tocar el botón.

Para conservar el efecto y cambiar el color del botón usa el atributo `app:backgroundTint` de la siguiente forma:

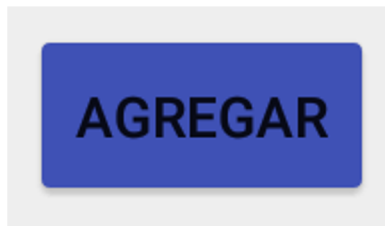




Cambiar el color del fondo

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    app:backgroundTint="@color/colorPrimary"
    android:text="@string/texto_agregar" />
</RelativeLayout>
```





Botón con Texto e Imagen

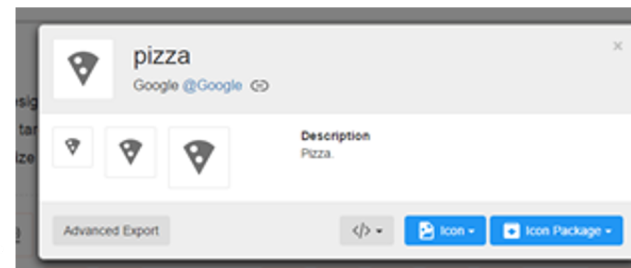
En la sección de “Atributos” vimos que existen atributos con la forma **android:drawable*** para alinear una imagen al texto de un botón.

Ejercicio

Alinear a la izquierda del botón “Ordenar” un icono asociado a pizzas.

Solución

1. Abre el layout activity_main.xml
2. Agrega un botón centrado en el padre.
3. Cambia el color de texto a blanco (@android:color/white)
4. usa un tinte de color rojo (#ef9a9a)
5. Y alinea a la izquierda el siguiente drawable de pizza

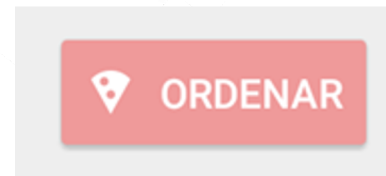


Botón con Texto e Imagen

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/ic_pizza"
    android:drawablePadding="8dp"
    android:id="@+id/boton"
    android:textColor="@android:color/white"
    app:backgroundTint="#ef9a9a"
    android:text="Ordenar"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Se usó para ubicar la imagen en la izquierda. Si compruebas con las sugerencias de Android tendrás varias posiciones: **derecha, abajo y arriba.**



```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawable
        android:drawableBottom
        android:drawableEnd
        android:drawableLeft
        android:drawableRight
        android:drawableStart
        android:drawableTint
        android:drawableTintMode
        android:drawableTop
        android:textCursorDrawable
```

<relat

Press Ctrl+Espacio to view tags from other namespaces



La clase ImageButton

ImageButton funciona exactamente cómo **Button**, solo que en lugar de traer un texto en su background, viene una imagen para especificar la acción.

Para cambiar la imagen de un ImageButton usa el atributo **android:src**. Obviamente su valor es un drawable.

El contorno del background se conserva como lo hemos visto hasta ahora, la diferencia está que en el centro se ubicará la imagen elegida en **src**.

Si quieres que el **background** por defecto desaparezca, asigna un color transparente o cambia el contenido con un list **drawable** (esto lo verás más adelante).



La clase ImageButton

Ejemplo

Cambiar la imagen de un image button con el icono de la app.

Solución

Lo primero es abrir el layout de la actividad y añadir un elemento `<ImageButton>` centrado en el relative layout.
El icono de la aplicación actual se encuentra en la referencia `@mipmap/ic_launcher`. Así que asigna este valor al atributo `android:src`.
Adicionalmente puedes el color del sistema `@android:color/transparent` sobre `android:background` para eliminar el contorno. Pero recuerda que esto elimina los efectos del Material Design.

Resultado

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/boton"  
    android:src="@mipmap/ic_launcher"  
    android:layout_centerVertical="true"  
    android:layout_centerHorizontal="true" />
```

Crear Botón Desde Android Studio

El Tab de Design de la parte inferior buscamos en la paleta de componentes "Buttons" y luego elegimos "Button" y lo arrastramos a la pantalla de diseño posicionándolo donde se quiera.



Manejar los Eventos de un Botón

Es de esperar que un botón dispare un evento al ser clickeado por un usuario, lo que permitirá ejecutar la acción a la que hace referencia dicho view.

Para procesar el evento existen varias formas de proceder.

Usar el Atributo `android:onClick`

Anteriormente cuando viste la clase `Button` se mencionó la existencia de `android:onClick()` para asignar un método que se ejecute cuando el usuario presione el botón.

Para ello se requiere que el método cumpla con las siguientes condiciones:

- Que sea público
- Que sea tipo void
- Que reciba un parámetro del tipo View
- Debe declararse en la actividad que usa el mismo layout

Manejar los Eventos de un Botón

The screenshot displays the Android Studio interface with the following components:

- Code Editor (Left):** Shows the `activity_main2.xml` file with the following XML code:

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:layout_width="match_parent"
6   android:layout_height="match_parent"
7   tools:context=".MainActivity">
8
9   <TextView
10     android:id="@+id/textView2"
11     android:layout_width="wrap_content"
12     android:layout_height="wrap_content"
13     android:text="Hello World!"
14     app:layout_constraintBottom_toBottomOf="parent"
15     app:layout_constraintLeft_toLeftOf="parent"
16     app:layout_constraintRight_toRightOf="parent"
17     app:layout_constraintTop_toTopOf="parent" />
18
19   <Button
20     android:id="@+id/button"
21     android:background="@color/colorPrimary"
22     android:layout_width="wrap_content"
23     android:layout_height="wrap_content"
24     android:layout_marginStart="156dp"
25     android:layout_marginTop="64dp"
26     android:onClick=""
27     app:layout_constraintBottom_toBottomOf="parent"
28     app:layout_constraintLeft_toLeftOf="parent"
29     app:layout_constraintRight_toRightOf="parent"
30     app:layout_constraintTop_toTopOf="@+id/textView2" />
31 </androidx.constraintlayout.widget.ConstraintLayout>
```
- Preview (Right):** Shows a visual representation of the layout with a button and a text view. Dimensions are indicated: 150dp for the button width and 64dp for the button height.
- Code Editor (Bottom):** Shows the Java implementation of the `onClick` event:

```
com.example.primerasesion.myapplication.MainActivity
public void goToActivity2(View view)
```




Usar escucha anonima OnClickListener

Otra forma es crear una instancia anónima de la interfaz **View.OnClickListener** para manejar los eventos del botón. Esto requiere usar el método **setOnClickListener()** para asignar el listener al botón y luego sobrescribir el controlador **onClick()** con las acciones a ejecutar.

Ejemplo: Iniciar otra actividad al presionar un botón

1. Lo primero es añadir otra actividad al proyecto actual que tienes abierto.
2. Lo siguiente es ir `activity_main2` (segunda activity de ejemplo creada) y obtener la instancia del botón que tenemos en `onCreate()`.
3. Luego invoca el método `setOnClickListener()` desde la instancia y como parámetro digita solamente "new O". Esto es suficiente para que Android Studio te recomiende la creación de un nuevo **OnClickListener**.

```
Button button = (Button)findViewById(R.id.button);
button.setOnClickListener(new O);
}

public void goToActivity2(View v) {
    Intent newIntent = new Intent(this, Activity2.class);

    // paso de parámetros
    newIntent.putExtra("name", "Misión TIC 2022");

    // inicialización de nueva actividad
    startActivity(newIntent);
}
```

View.OnClickListener (android.view.View)

- OutOfMemoryError (java.lang)
- Override (java.lang)
- Object (java.lang)
- OnCreateContextMenuListener (android.view.View)
- OnApplyWindowInsetsListener (android.view.View)
- OnAttachStateChangeListener (android.view.View)
- OnCapturedPointerListener (android.view.View)
- OnContextClickListener (android.view.View)
- OnDragListener (android.view.View)
- OnFocusChangeListener (android.view.View)
- OnScrollListener (android.view.View)

View.OnClickListener
Kotlin > Java

public static interface
View.OnClickListener

Interface definition for a callback to be
invoked when a view is clicked.

< Android API 29 Platform (2) >

"OnClickListener in View" on
developer.android.com >



Usar escucha anonima OnClickListener

Al presionar ENTER o clickear la sugerencia, Android Studio creará la escucha anónima junto a la implementación del controlador onClick():

```
Button boton = (Button) findViewById(R.id.button);  
boton.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
  
    }  
});
```

Ahora dentro del controlador crea un nuevo Intent para iniciar la actividad Main2Activity. Seguido invoca startActivity() para hacer efectivo el inicio.

```
Button boton = (Button) findViewById(R.id.button);  
boton.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        Intent i = new Intent(MainActivity.this,  
Main2Activity.class);  
        startActivity(i);  
    }  
});
```



El futuro digital
es de todos

MinTIC

Tema 4 - Checkbox

Cómo usar el control Checkbox

Recibir actualizaciones



Un Checkbox es un botón de dos estados (marcado, no marcado) que actúa como control de selección (los radio buttons y switches también pertenecen a esta categoría) ante los usuarios. Lo que permite elegir una o varias opciones de un conjunto.



Cómo usar el control CheckBox

Por defecto su color será el mismo de la propiedad **android:colorAccent** en el tema de la aplicación.

La representación en Java de este control es con la clase **CheckBox**

Esto quiere decir que debes usar la etiqueta **<CheckBox>** en tus layouts para crearlos en la vista.

Abre el layout `activity_main.xml` y agrega un nuevo checkbox como muestra el siguiente código

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:layout_marginEnd="144dp"
    android:layout_marginBottom="148dp"
    android:text="Recordarme"
    android:onClick="loguearCheckbox"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />
```



Obtener Valor Del CheckBox

Si quieres saber en qué estado se encuentra actualmente el CheckBox utiliza el método `isChecked()`.

El retorno es de tipo boolean, donde:

- True ➤ checked
- False ➤ unchecked.

Obtener el estado de un checkbox y mostrarlo en un Toast al pulsarlo

1. Agregar el checkbox como se mencionó anteriormente.
2. Abrir **MainActivity.java** y en el **OnCreate** buscar el **checkbox** creado.
3. Declarar el método **loguearCheckbox()**. En su interior llama el método estático **Toast.makeText()** y loguea el estado del check box.
4. Podemos agregar al checkbox para cuando se dé click en el que llame al **método loguearCheckbox()**, esto se puede hacer llamando al **setOnClickListener** o directamente desde el layout.

Obtener Valor Del CheckBox

```
activity_main2.xml | MainActivity.java | MainActivity.java | activity_main.xml | Preview
```

```
1 private CheckBox checkBoxTest;
2
3 @Override
4 protected void onCreate(Bundle savedInstanceState) {
5     super.onCreate(savedInstanceState);
6     setContentView(R.layout.activity_main);
7
8     Button button = (Button) findViewById(R.id.button);
9     button.setOnClickListener(new View.OnClickListener() {
10         Intent newIntent = new Intent(packageContext, MainActivity.class);
11
12         // paso de parámetros
13         newIntent.putExtra("name", "Hello Activity 2");
14
15         // inicialización de nueva activity
16         startActivity(newIntent);
17     });
18
19 2 checkBoxTest = findViewById(R.id.checkBox);
20
21 public void loguearCheckBox(View v) {
22     String s = "Estado: " + (checkBoxTest.isChecked() ? "Marcado" : "No Marcado");
23     Toast.makeText(context, this, s, Toast.LENGTH_LONG).show();
24 }
25
26 public void goToActivity2(View view) {
27     Intent newIntent = new Intent(packageContext, MainActivity.class);
28
29     // paso de parámetros
30     newIntent.putExtra("name", "Hello Activity 2");
31
32     // inicialización de nueva activity
33     startActivity(newIntent);
34 }
35 }
```

```
17 app:layout_constraintRight_toRightOf="parent"
18 app:layout_constraintTop_toTopOf="parent"
19 app:layout_constraintTop_toTopOf="parent" />
20
21 <Button
22     android:id="@+id/button"
23     android:background="@color/colorPrimary"
24     android:layout_width="wrap_content"
25     android:layout_height="wrap_content"
26     android:layout_marginStart="136dp"
27     android:layout_marginTop="64dp"
28     android:onClick="goToActivity2"
29     android:text="ACTIVIDAD 2"
30     app:layout_constraintStart_toStartOf="parent"
31     app:layout_constraintTop_toTopOf="@+id/textView2" />
32
33 <CheckBox
34     android:id="@+id/checkBox"
35     android:layout_width="wrap_content"
36     android:layout_height="wrap_content"
37     android:layout_centerHorizontal="true"
38     android:layout_centerVertical="true"
39     android:layout_marginEnd="144dp"
40     android:layout_marginBottom="144dp"
41     android:text="Marcado"
42     android:onClick="loguearCheckBox"
43     app:layout_constraintBottom_toBottomOf="parent"
44     app:layout_constraintEnd_toEndOf="parent" />
45
46 </androidx.constraintlayout.widget.ConstraintLayout>
```

Preview

Hello World

ACTIVIDAD 2

Random



El futuro digital
es de todos

MinTIC

Tema 5 - TextView



Cómo usar el control TextView

El TextView En Android **es un widget que muestra texto al usuario** como su nombre lo sugiere. Claramente esto lo hace ser uno de los views más usados en interfaces de usuario para proyectar cabeceras, títulos, texto informativo, etiquetas y muchos otros.



Tamaño de 14sp

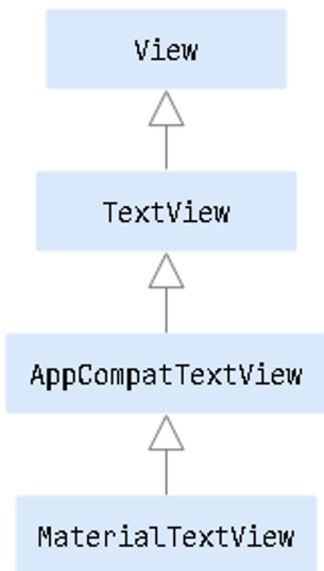
Tamaño de 16sp

Tamaño de 20sp

Tamaño de 24sp



La Clase TextView



TextView es una de las directas descendientes de la clase View. Al ser la abstracción responsable de contener texto, varios de los elementos gráficos del paquete android.widget, como Button y EditText, reutilizan esta naturaleza.

Por otro lado, para darle soporte a versiones anteriores de Android de las nuevas capacidades que se van introduciendo al TextView, se creó la clase **AppCompatActivity**.

También tienes a disposición la clase **MaterialTextView** para aplicar los temas de material design sobre tus text views por defecto.

```
<TextView  
    android:id="@+id/hello_world_text"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello World!" />
```

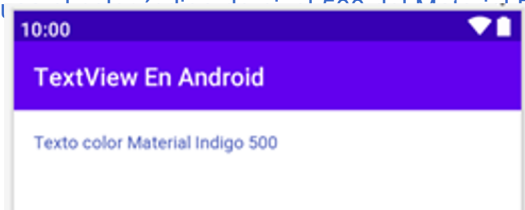


Atributos Del TextView

Cambia el color del texto a través del atributo **android:textColor**. Este recibe la referencia a un recurso de color o valores RGB en variaciones "rgb", "argb", "rrggbb", o "aarrggbb":

```
<TextView
    android:id="@+id/text_color"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#3F51B5"
    android:text="Texto color Material Indigo 500"
/>
```

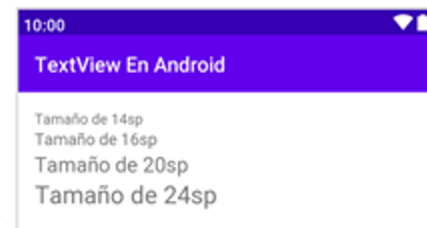
Este ejemplo muestra el resultado de un TextView con el atributo android:textColor="3F51B5" en Material Design:



El atributo **android:textSize** determina el tamaño del texto y se recomienda asignarle valores en pixeles escalados o sp. Sin embargo puedes usar medidas en px, dp, in y mm:

```
<TextView
    android:id="@+id/text_size_24sp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:text="Tamaño de 24sp" />
```

Este ejemplo junto a las dimensiones de 14sp, 16sp y 20sp se verían así:



También es posible usar un recurso de dimensiones a través de la navegación @dimen/nombre_dimension.

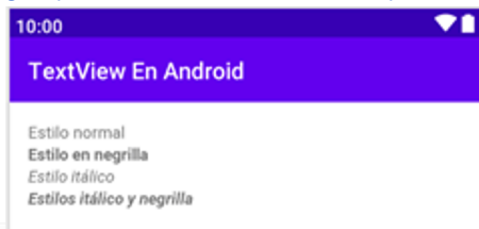


Atributos Del TextView

Asigna uno de los siguientes estilos o combinaciones de ellos: normal, bold y italic. El valor por defecto es normal y si deseas combinarlos usa el símbolo '|':

```
<TextView  
    android:id="@+id/text_style_combination"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Estilos itálico y negrilla"  
    android:textStyle="bold|italic" />
```

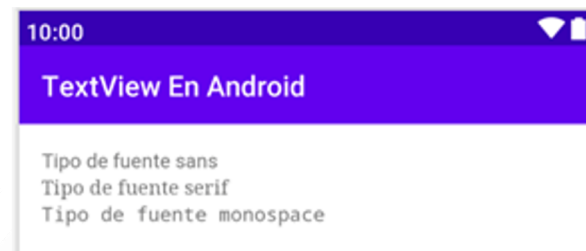
En la siguiente imagen puedes ver todos los estilos posibles:



El atributo **android:typface** acepta las siguientes constantes para especificar el estilo de fuente del TextView: normal, sans, serif y monospace:

```
<TextView  
    android:id="@+id/typface_monospace"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Tipo de fuente monospace"  
    android:typface="monospace" />
```

La siguiente imagen muestra la diferencia entre las formas y trazos que se usan en cada estilo tipográfico:





Convertir URLs En Links Clickeables

Si deseas habilitar la detección de patrones que coincidan con esquemas como: correos, urls, teléfonos, entre otros; entonces, aplica el atributo **android:autoLink**.

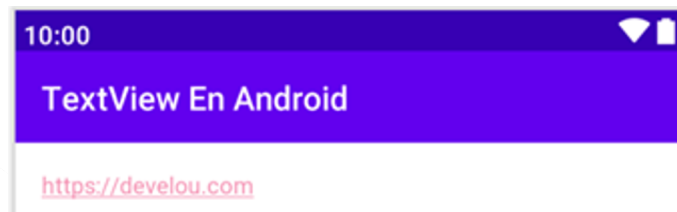
Por ejemplo, el siguiente TextView genera un link clickeable hacia Develou

```
<TextView  
    android:id="@+id/autolink_url"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="https://develou.com"  
    android:textColorLink="#F48FB1"  
    android:autoLink="web" />
```

Los siguientes son los valores que puedes asignar:

- **all**: Todos los patrones
- **email**: Direcciones de correo
- **none**: Ninguno (por defecto)
- **phone**: Teléfonos
- **web**: URLs web

También puedes usar **android:textColorLink** para modificar el color de creación del link.

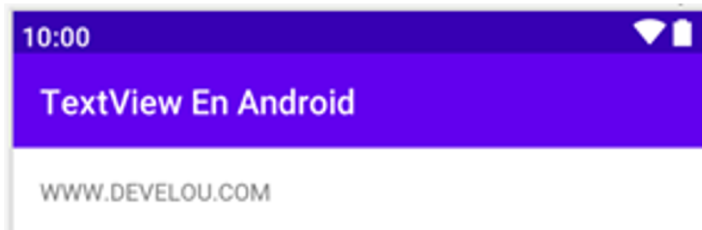




Convertir Texto A Mayúsculas

```
<TextView  
    android:id="@+id/text_all_caps"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="www.develou.com"  
    android:textAllCaps="true" />
```

Esto genera la siguiente presentación del texto:



En el caso que desees convertir toda la entrada de texto a mayúsculas, aplica true al atributo **android:textAllCaps**.

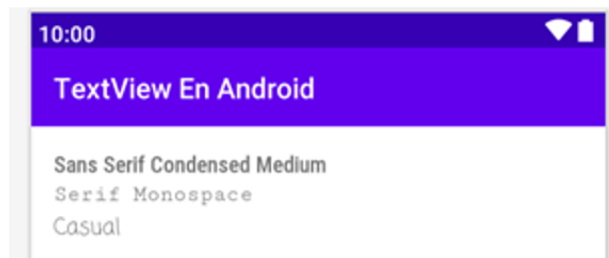


Familia Tipográfica

Define el tipo que será usado de una fuente declarada en el sistema o en tus recursos de fuente (res/font).

```
<TextView
    android:id="@+id/font_family1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:fontFamily="sans-serif-condensed-medium"
    android:text="Sans Serif Condensed Medium " />
```

En el sistema encontrarás familias prefabricadas para generar variaciones con las fuentes del sistema.

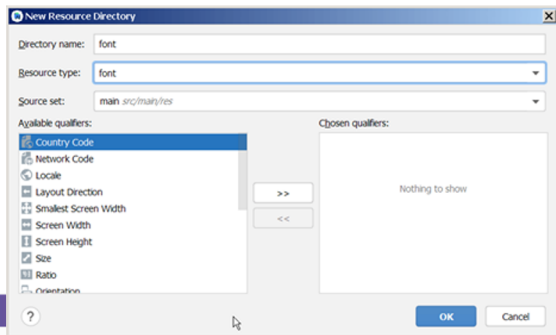




Agregar tu propia fuente

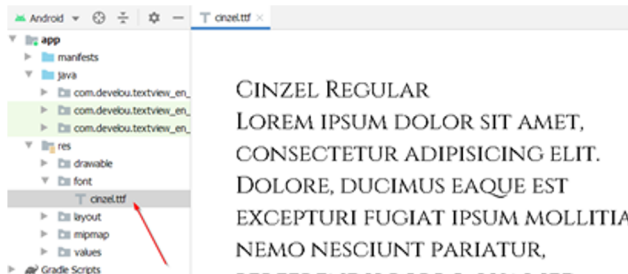
Paso 1

Crea la carpeta de recursos tipo fuente, dando click derecho en **res** y seleccionando **Android Resource Directory**. Luego asegúrate de seleccionar el tipo **font**.



Paso 2

Añade al directorio **font** tus archivos de fuente. En este ejemplo añadiremos la fuente Cinzel desde Google Fonts

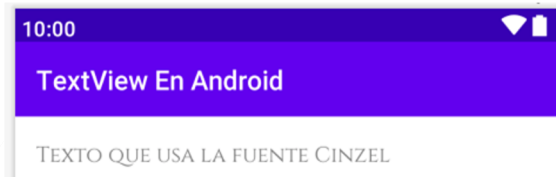


CINZEL REGULAR
LOREM IPSUM DOLOR SIT AMET,
CONSECTETUR ADIPISICING ELIT.
DOLORE, DUCIMUS EAQUE EST
EXCEPTURI FUGIAT IPSUM MOLLITIA
NEMO NESCIUNT PARIATUR,
PERFERENDIS PORRO QUAS SED.
HARUM NIHIL QUIDEM VENIAM!

Paso 3

Ve al **TextView** y referencia a la fuente con la dirección **@font/cinzel**. De esta se aplicará la fuente al texto:

```
<TextView  
    android:id="@+id/custom_font"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textSize="16sp"  
    android:fontFamily="@font/cinzel"  
    android:text="Texto que usa la fuente Cinzel" />
```





Apariencia De Texto

El atributo **android:textAppearance** permite definir en conjunto el color de texto, tipo de letra, tamaño y estilo de texto. Normalmente usarás este atributo para asignar las apariencias prefabricadas en los estilos del sistema.

Estas concuerdan con las escalas de tipografía definidas en la guía del Material Design. Y tan solo es seleccionarlas del menú emergente cuando tipeas la escala.

```
<TextView
    android:id="@+id/headline_4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Ejemplos"
    android:textAppearance="@style/Headline" />
```

nearLayout>

- @style/TextAppearance.AppCompat.Headline
- @style/TextAppearance.MaterialComponents.Headline6
- @style/TextAppearance.MaterialComponents.Headline1
- @style/TextAppearance.MaterialComponents.Headline2
- @style/TextAppearance.MaterialComponents.Headline3
- @style/TextAppearance.MaterialComponents.Headline4
- @style/TextAppearance.MaterialComponents.Headline5

nearLayout > TextView

Press Intro to insert, Tabulador to replace



Apariencia De Texto

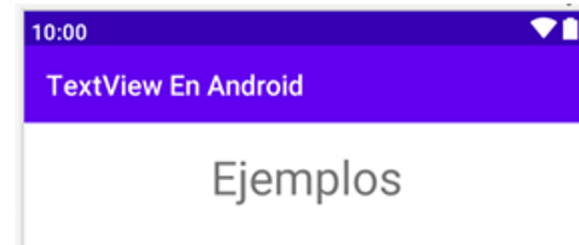
Por ejemplo, el siguiente texto usa la apariencia **TextAppearance.MaterialComponents.Headline4**:

```
<TextView  
    android:id="@+id/headline_4"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    android:text="Ejemplos"  
    android:textAppearance="@style/TextAppearance.  
MaterialComponents.Headline4" />
```

El resultado es la aplicación de los siguientes valores:

```
TextAppearance.MaterialComponents.Headline4:  
    android:android:fontFamily = sans-serif  
    android:android:letterSpacing = 0.00735294118  
    android:android:textAllCaps = false  
    android:android:textSize = 34sp  
    android:android:textStyle = normal  
    fontFamily = sans-serif
```

El texto se proyectará de la siguiente forma:





El futuro digital
es de todos


MinTIC

Tema 6 - ImageView

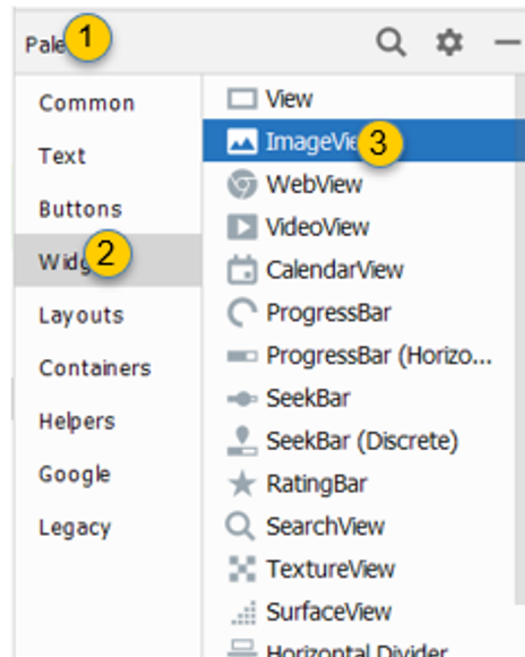


Cómo usar el control ImageView

Este componente se usa en Android para mostrar imágenes en la interfaz de tus aplicaciones, a partir de recursos **drawables** o elementos de la clase Bitmap.

Desde Android Studio podemos agregar una imagen desde el editor de layouts, así: 

Luego arrastra y suelta en el lienzo.



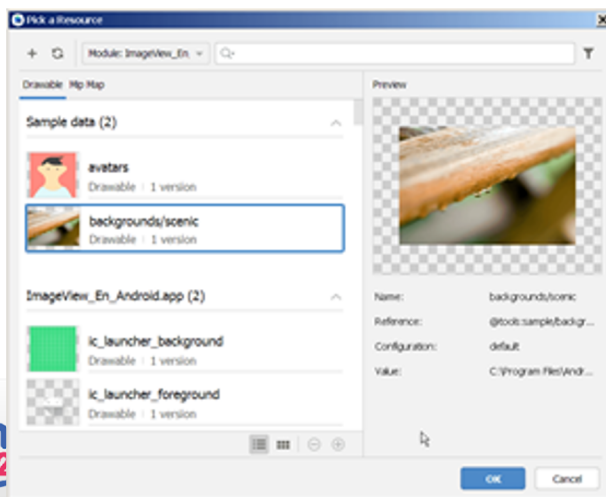
Cómo usar el control ImageView

En seguida, se te mostrará una ventana para que elijas el recurso que será asignado al **ImageView**. Puedes elegir entre **drawables** de ejemplo para **avatars** o **backgrounds** (antecedidos por la dirección **@tools:sample**), los de tu proyecto o los del sistema.

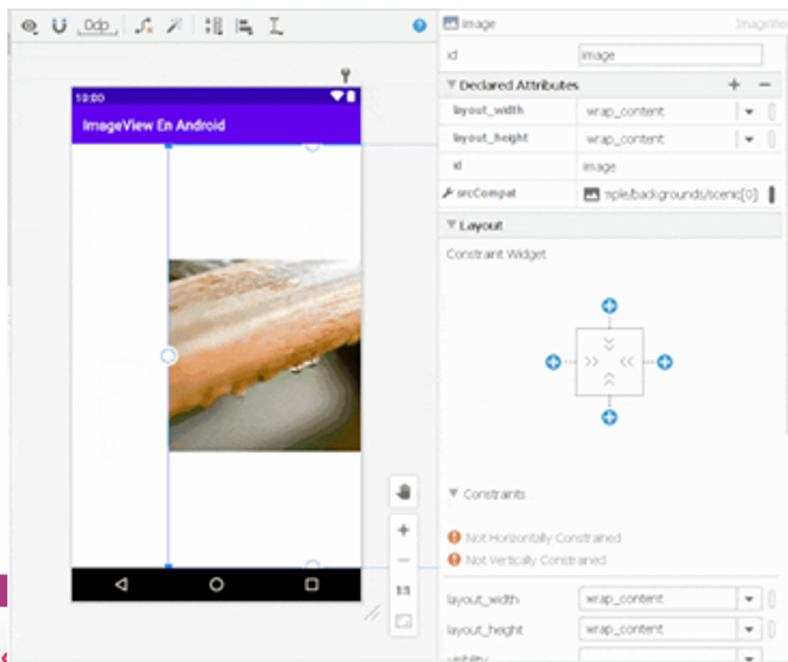
Elige la opción de backgrounds de ejemplo por el momento y presiona OK para agregar la etiqueta XML al layout. Esto producirá una definición similar a la siguiente:

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:srcCompat="@tools:sample/backgrounds/scenic" />
```

Claramente si estás dentro de un **ConstraintLayout**, la imagen aparecerá desconectada de restricciones y se verá desparramada sin rumbo sobre el lienzo.



En este ejemplo conectaremos sus límites con los del padre y si tienes un TextView por defecto, como es nuestro caso, entonces limita el borde inferior de la imagen con el superior del texto.



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<ImageView
    android:id="@+id/image"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf="@+id/textView"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@tools:sample/backgrounds/scenic" />
```

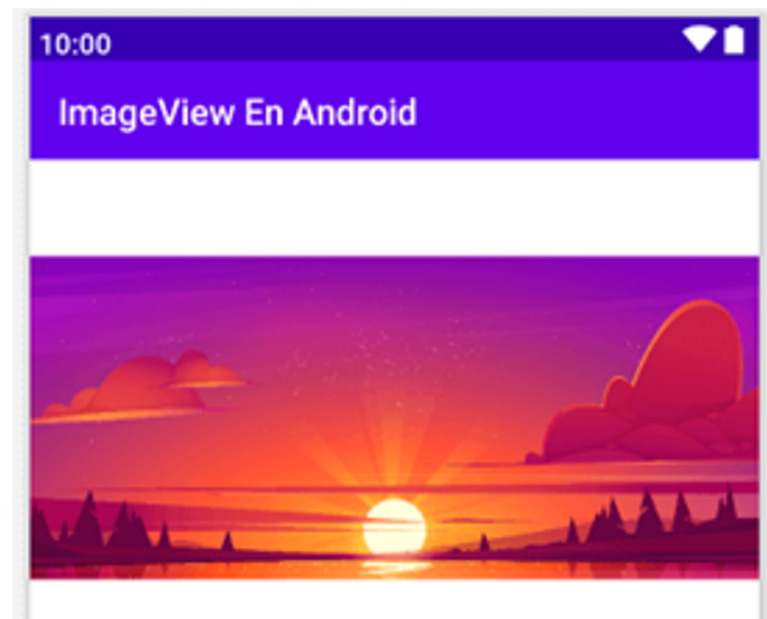
```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Asignar Drawable A ImageView

Sin embargo, el atributo **tools:srcCompat** es sólo una herramienta para renderizar la imagen de ejemplo en el diseño, por lo que al correr el aplicativo no verás la imagen.

Asignar realmente un drawable consiste en aplicar una dirección al atributo **android:src** o **app:srcCompat** en caso de que el drawable sea un vector:

```
app:srcCompat="@drawable/sunset"
```





Asignar Drawable A ImageView

También es posible asignar el valor a este atributo en Java o Kotlin a través de los siguientes métodos:

setImageBitmap(): Asigna una instancia Bitmap como fuente de imagen

setImageDrawable(): Asigna como fuente una instancia de la clase Drawable

setImageResource(): Asigna como fuente un recurso drawable

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        ...  
        ImageView imageViewTest = findViewById(R.id.imageView);  
        imageViewTest.setImageResource(R.drawable.sunset);  
    }  
}
```




Asignar Drawable A ImageView

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button button = (Button) findViewById(R.id.button);
    button.setOnClickListener((view) -> {
        Intent newIntent = new Intent( packageContext: MainActivity.this, Main2Activity.class);

        // paso de parámetros
        newIntent.putExtra( name: "name", value: "Hello Activity 2");

        // inicialización de nueva activity
        startActivity(newIntent);
    });

    checkBoxTest = findViewById(R.id.checkBox);

    ImageView imageViewTest = findViewById(R.id.imageView);
    imageViewTest.setImageRe

}

public void logearCheckbox(View v) {
    String s = "Estado: " + (checkBoxTest.isChecked()) ? "Marcado" : "No
    Toast.makeText( context: this, s, Toast.LENGTH_SHORT).show();
}

public void goToActivity2(View view) {
    Intent newIntent = new Intent( packageContext: this, Main2Activity.cl

}

// paso de parámetros
MainActivity -> onCreate()
```

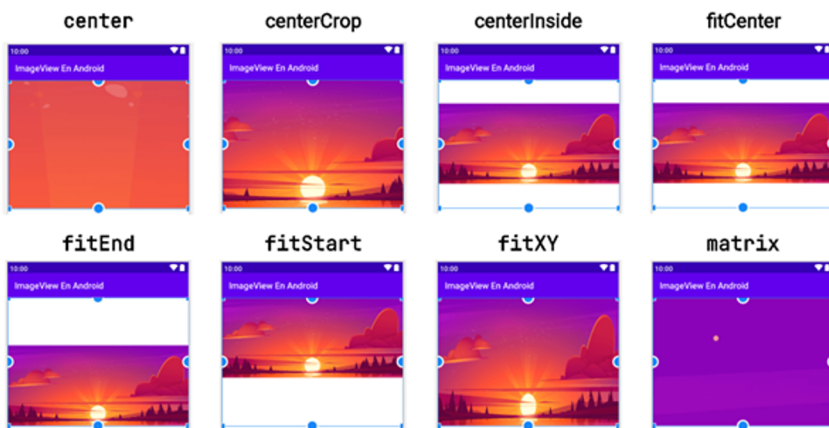
setImageResource(int resId) void
Press ↵ to choose the selected (or first) suggestion and insert a dot afterwards. >>>

android.widget.ImageView
public void setImageResource(@DrawableRes int resId)
External annotations: @android.support.annotation.DrawableRes
The following documentation urls were ch
<http://developer.android.com/reference/android/widget/ImageView>
<http://developer.android.com/reference/android/widget/ImageView>
[Edit API docs paths](#)

< Android API 29 Platform (2) >
`setImageResource(int)` on developer.android.com >
`setImageResource(int)` on developer.android.com >

Modificar escala de la imagen

Usa el atributo **android:scaleType** para controlar cómo redimensionar o mover el contenido de la imagen para que coincida con el rectángulo del **ImageView**. La siguiente es una tabla con las constantes que puedes asignarle a este atributo:



Constante	Descripción
<code>center</code>	Centra la imagen en el view sin realizar escalado.
<code>centerCrop</code>	Escala el ancho y alto de la imagen, manteniendo la relación de aspecto. De tal forma que ambas dimensiones sean iguales o mayores a las del view. Luego es centrada.
<code>centerInside</code>	Igual que <code>centerCrop</code> , solo que el escalado hace que las dimensiones de la imagen sean iguales o menores a las del view.
<code>fitCenter</code>	Computa un matrix que mantenga el ratio de la imagen, asegurándose que encaje en el view por completo. Al menos uno de los ejes (X o Y) será ajustado y luego se centrará el resultado final.
<code>fitEnd</code>	Igual que <code>fitCenter</code> , solo que se ajusta la imagen hacia el borde inferior derecho del view
<code>fitStart</code>	Igual que <code>fitCenter</code> , solo que se ajusta la imagen hacia el borde superior izquierdo del view
<code>fitXY</code>	Escala el alto y ancho independientemente para que coincidan con el tamaño del view. Claramente no se conservará el ratio.
<code>matrix</code>	Usa una matrix de transformación que proveas a través de <code>setImageMatrix(Matrix)</code>



El futuro digital
es de todos

MinTIC

Tema 7 - EditText



Cómo usar el control EditText

Un **EditText** es un **TextView** cuya apariencia ha sido modificada para actuar como campo de texto, donde el usuario puede editar su contenido para especificar datos en una aplicación Android.



Texto de entrada

Un EditText es la **expansión de un TextView con la capacidad de editar su contenido para recibir texto por parte del usuario**. Visualmente estos proyectan una línea inferior del color del acento del tema y un texto auxiliar llamado hint que representa el contenido asociado al view.



Cómo usar el control EditText

Ejemplo

Abre el archivo **activity_main.xml** y agrega el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
  <EditText
    android:id="@+id/campo_texto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:hint="Texto de entrada" />
</RelativeLayout>
```

En la anterior definición se centra un EditText en el RelativeLayout,

Cuyo ancho se ajusta al padre y el alto al contenido. Además se usa el texto auxiliar «**Text de entrada**» en el atributo android:hint.



Cómo usar el control EditText



Esto es lo que verás cuando ves la ventana
Preview

Si deseas personalizar un EditText basate en [los atributos de TextView](#)



Obtener Texto Del EditText

Para retornar el valor de texto de un EditText usa el método **getText()**.

Este no retorna directamente un objeto String, si no Editable. La cual es una interfaz de texto dinámico y configurable.

Sin embargo al usar **toString()** es posible obtener la cadena plana.



Añadir un campo de texto y agrega un botón por debajo, que al ser clickeado muestre en el logcat el valor actual.

1. Abre `actividad_principal.xml`

- Modifica el layout para que el botón aparezca por debajo del **EditText**.
- Luego asigna un manejador de clicks al botón con el atributo **onClick**.
- El nombre del manejador será **verValor**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/campo_texto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:hint="Teléfono"
        android:inputType="phone" />

    <Button
        android:id="@+id/boton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/campo_texto"
        android:layout_centerHorizontal="true"
        android:onClick="verValor"
        android:text="Guardar" />

</RelativeLayout>
```




Añadir un campo de texto y agrega un botón por debajo, que al ser clickeado muestre en el logcat el valor actual.

2. Abre `ActividadPrincipal.java` para definir el método `verValor()`.
 - a. Dentro de este obtén la instancia del view con la referencia `R.id.campo_texto`
 - b. Y loguea el resultado de `getText()`.
3. Ejecuta la app y presiona el botón luego de escribir algún número.

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;

public class ActividadPrincipal extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.actividad_principal);
    }

    public void verValor(View v) {
        EditText campoTexto = findViewById(R.id.campo_texto);
        Log.d("Valor ET", campoTexto.getText().toString());
    }
}
```

```
03-01 12:02:02.127 13892-13892/com.herprogramacion.camposdetexto D/Valor ET: 4566678
03-01 12:04:26.681 13892-13892/com.herprogramacion.camposdetexto D/Valor ET: 1
03-01 12:09:18.963 13892-13892/com.herprogramacion.camposdetexto D/Valor ET: 311 621 7878
```



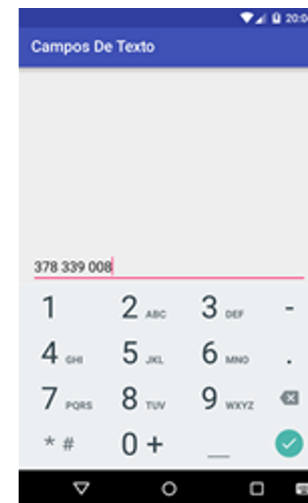
Tipos de Entrada en un Campo de Texto

El atributo **android:inputType** condiciona la entrada de texto al usuario para **ingresar caracteres acordes al requerimiento del EditText**.

Además de evitar que se escriban dichos caracteres, este atributo determina el tipo de teclado virtual que aparecerá ante el usuario y determina otros tipos de comportamientos.

```
<EditText
    android:id="@+id/campo_texto"
    android:layout_width="match_parent"
    android:inputType="phone"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:hint="Teléfono" />
```

Si ejecutas la app verás cómo el teclado se reduce a un keyboard tipo teléfono





Tipos de Entrada en un Campo de Texto

Constante	Descripción
text	Recibe texto plano simple
textPersonName	Texto correspondiente al nombre de una persona
textPassword	Protege los caracteres que se van escribiendo con puntos
numberPassword	Contraseña de solo números enmascarada con puntos
textEmailAddress	Texto que será usado en un campo para emails
phone	Texto asociado a un número de teléfono
textPostalAddress	Para ingresar textos asociados a una dirección postal

Constante	Descripción
textMultiLine	Permite múltiples líneas en el campo de texto
time	Texto para determinar la hora
date	Texto para determinar la fecha
number	Texto con caracteres numéricos
numberSigned	Permite números con signo
numberDecimal	Para ingresar números decimales

Especificar la Cantidad Máxima de Caracteres con maxLength



El futuro digital
es de todos

MinTIC

Para forzar el tamaño del texto que recibirá el EditText usa el atributo **android:maxLength**.
Especifica un número entero positivo para determinar cuántos caracteres podrá haber.
Este atributo es de gran utilidad cuando las reglas de negocio indican restricciones a las entradas del usuario.

Ejemplo

Solución

Resultado

Crear **EditText** para el nombre del usuario. Este no debe tener más de 8 caracteres.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/nombre_usuario"
    android:inputType="text"
    android:hint="Nick"
    android:maxLength="8"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
```

Agrega **maxLength** con el valor de 8 al campo:

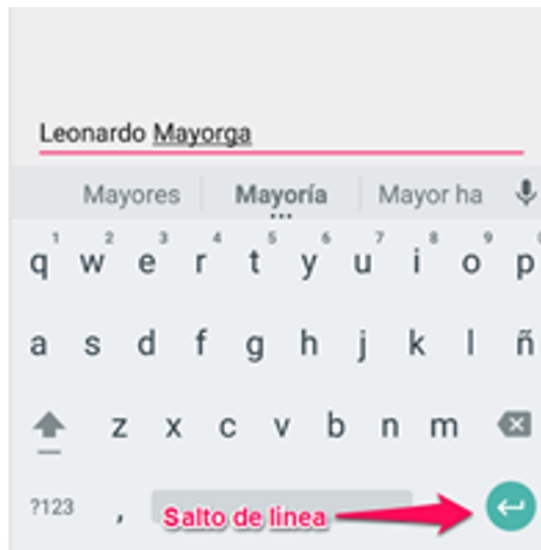
Si pruebas el resultado, el campo no te permitirá ingresar más de 8 caracteres.



EditText con una Línea

Reduce la capacidad del campo de texto a una sola línea con el atributo **android:singleLine**.

De lo contrario el EditText aceptará múltiples líneas en su contenido y el teclado virtual usará como tecla de acción el salto de línea en vez de la confirmación.



El valor por defecto es false, pero si usas un valor para textInput, entonces el valor de singleLine será true automáticamente.



EditText con una Línea

Ejemplo

Añadir un campo de texto para el nombre del conductor.

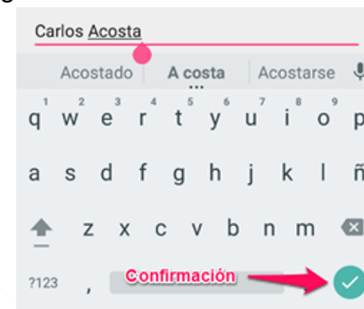
Solución

Añade el valor true a **android:singleLine** del editor.

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/nombre_conductor"  
    android:hint="Nombre del conductor"  
    android:singleLine="true"  
    android:layout_centerVertical="true"  
    android:layout_alignParentRight="true"  
    android:layout_alignParentEnd="true" />
```

Resultado

El resultado esperado será el siguiente:



La tecla de confirmación permitirá cerrar el teclado indicando que estás satisfecho con la edición.



Validación de datos



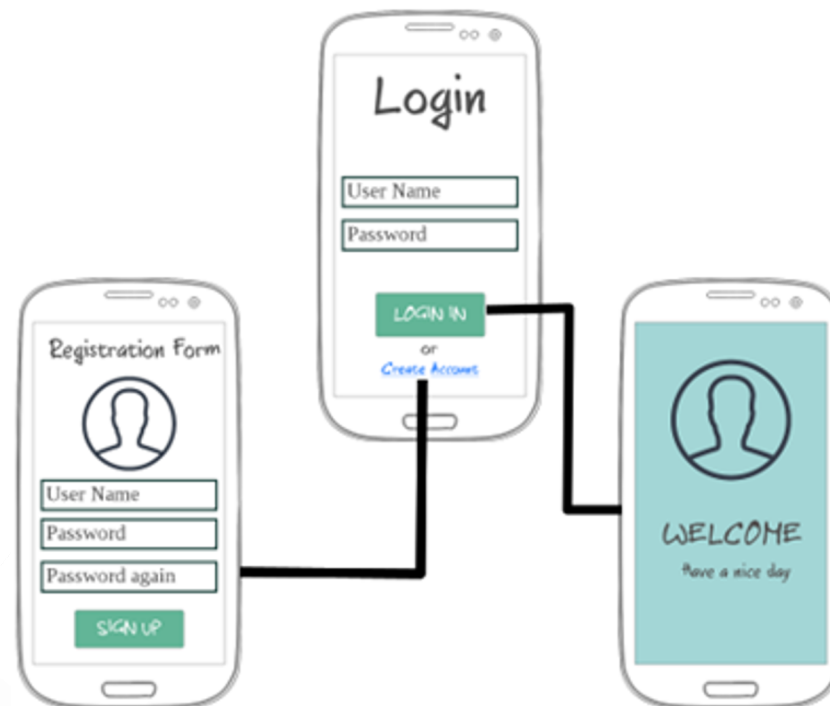
*"No
reinventes
la rueda"*

A la hora de desarrollar aplicaciones siempre encontramos **componentes que se repiten**, tal es el caso del módulo que permite validar un usuario para así conceder o restringir su acceso al contenido de una aplicación, debido a esta circunstancia Android Studio nos da la opción de **crear una actividad prescrita para proporcionar esta funcionalidad**, evitándose así caer en una frase muy popular entre desarrolladores **"no reinventes la rueda"**, pero está claro que para utilizar una rueda ya elaborada hay que conocer cómo funciona, es por ello que **hablaremos del funcionamiento de un Login Activity**.

Validación de datos

En esta imagen se esboza a grandes rasgos el **comportamiento de la aplicación**, además de incluir otros componentes que ayudarán a entender el uso de esta funcionalidad en un entorno de trabajo real.

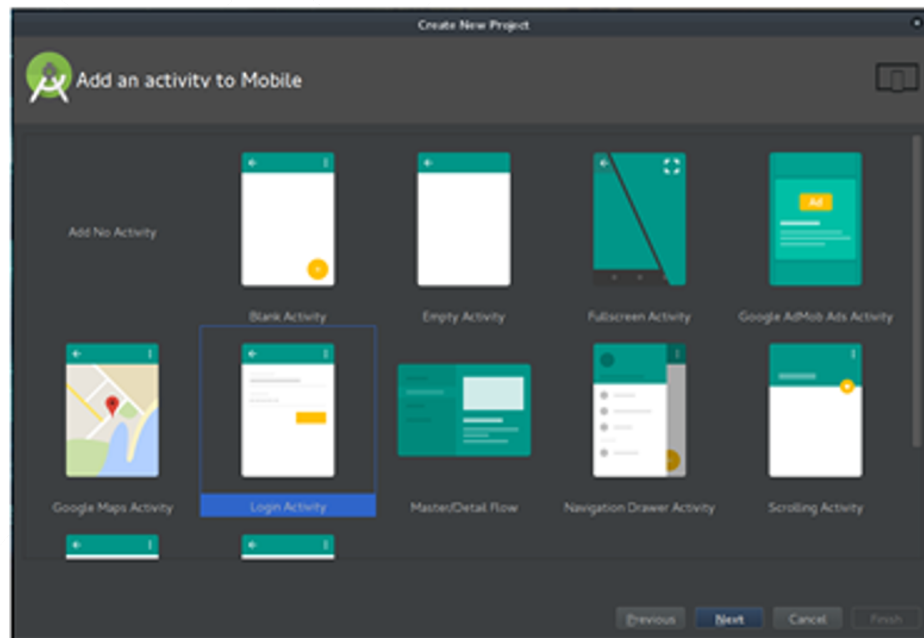
A continuación veremos el proceso de creación





Creación y Diseño del Login Activity

Primero vamos a crear nuestro proyecto en Android Studio, para ello lo abrimos y seleccionamos la opción **Start a new Android Studio project**, avanzamos por las ventanas de configuración hasta llegar a una en la que se desprenden varias opciones para iniciar nuestra **Activity**, entre las que encontramos:





Creación y Diseño del Login Activity



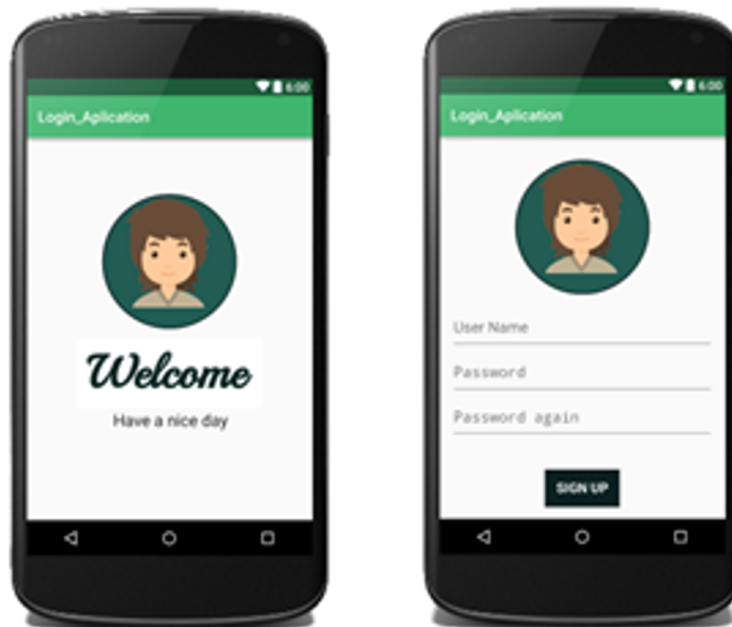
Una vez creado nuestro Login Activity, analizamos la interfaz **activity_login.xml** que viene por defecto y procedemos a modificarla para darle una mejor apariencia



Creación y Diseño del Login Activity

Además del primer Activity que modificamos ahora vamos a crear los correspondientes para el **Mensaje de Bienvenida** y el **Registro del Usuario** presentados en la siguiente Imagen.

Ahora que tenemos lista toda la interfaz de nuestra aplicación procederemos a analizar el código generado por nuestro **Login Activity**.



Análisis del Funcionamiento de la Aplicación



mEmailView: Añadiendo las funcionalidades de autocompletado a la caja de texto.

```
5
6 mEmailView = (AutoCompleteTextView) findViewById(R.id.email);
7
8 /**Cargar la Función para el Autocompletado de Emails. */
9 populateAutoComplete();
10
```

mPasswordView: Añadiendo el evento *ENTER* en la caja de texto password, si se produce el evento se lanza la función *attemptLogin* para validar el formulario.

```
5
6 mPasswordView = (EditText) findViewById(R.id.password);
7 /**Adiciona la escucha del evento ENTER sobre la caja de Texto de la contraseña*/
8 mPasswordView.setOnEditorActionListener(new TextView.OnEditorActionListener() {
9     @Override
10     public boolean onEditorAction(TextView textView, int id, KeyEvent keyEvent) {
11         if (id == R.id.login || id == EditorInfo.IME_NULL) {
12             attemptLogin();
13             return true;
14         }
15         return false;
16     }
17 });
18
```

Análisis del Funcionamiento de la Aplicación



mEmailSignInButton: Añadiendo el evento *Click* sobre el boton Sign In, si se produce el evento se lanza la función `attemptLogin` para validar el formulario.

```
5
6 Button mEmailSignInButton = (Button) findViewById(R.id.email_sign_in_button);
7 mEmailSignInButton.setOnClickListener(new OnClickListener() {
8     @Override
9     public void onClick(View view) {
10        attemptLogin();
11    }
12 });
13
```

registForm: Añadiendo el evento *Click* sobre el textView Create Account, si se produce el evento se lanza un nuevo Activity para el registro de un nuevo usuario.

```
5
6 registForm = (TextView) findViewById(R.id.createAccount);
7 registForm.setOnClickListener(new OnClickListener() {
8     @Override
9     public void onClick(View v) {
10        Intent i = new Intent(LoginActivity.this, Registration.class);
11        startActivity(i);
12    }
13 });
14
```



El futuro digital
es de todos

MinTIC

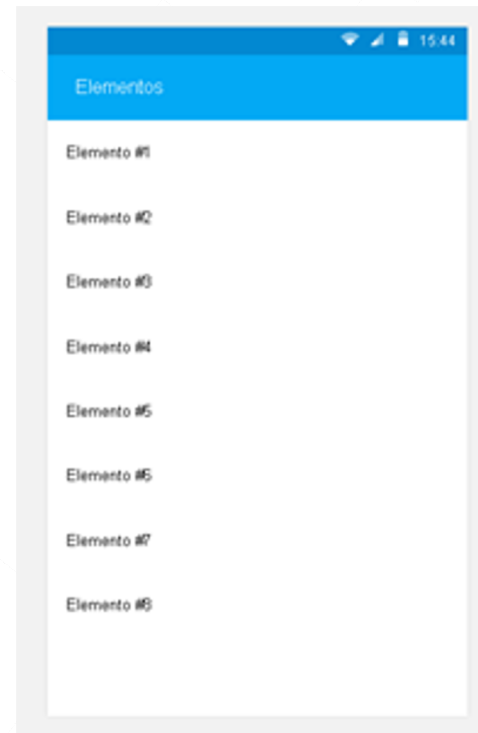
Tema 8 - ListView



Como usar el control ListView



La clase que representa una lista vertical en el **API de Android** se llama ListView. Esta clase viene preparada para recibir los ítems que desplegará en la interfaz, facilitando al programador la implementación de sus características y comportamientos.





Como usar el control ListView



Si en algún momento los ítems que contiene dificultan la visualización total en la actividad de la aplicación, automáticamente implementará **scrolling** para que el usuario pueda desplegar los elementos ocultos.

Estructuralmente un ListView contiene un View específico por cada fila. También se compone de un ScrollView, el cual permite generar el desplazamiento vertical por si se agota la pantalla para nuestros elementos.



Añadir ListView al layout

En la definición XML las listas se agregan con el elemento `<ListView>`. Debido a que esta clase extiende de `ViewGroup`, es posible que lo uses como nodo raíz de un `layout`.

```
<ListView  
    android:id="@+id/leads_list"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

Si quieres cambiar aspectos en su contenido o comportamiento puedes usar atributos XML como los siguientes:

Atributo	Función
<code>android:divider</code>	Drawable o color para representar el divisor entre ítems. Por defecto verás una línea horizontal de color gris claro. Usa el valor <code>@null</code> si no quieres que aparezca.
<code>android:dividerHeight</code>	Altura del divisor
<code>android:entries</code>	Aquí puedes poner la referencia de un array de strings de tus recursos para poblar automáticamente la lista sin usar adaptadores.
<code>android:footerDividersEnabled</code>	Habilita o deshabilita el divisor que va antes de un elemento especial llamado «footer», el cuál va al final de la lista.
<code>android:headerDividersEnabled</code>	Similar a <code>android:footerDividersEnabled</code> , solo que esta vez se refiere al divisor de un elemento especial llamado «header» que va al inicio de la lista.

LLenar Lista Con Un Adaptador

Un **adaptador** es un objeto que comunica a un ListView los datos necesarios para crear las filas de la lista.

Es decir, conecta la lista con una fuente de información como si se tratase de un adaptador de corriente que alimenta a un televisor.



Además de proveer la información, también genera los Views para cada elemento de la lista

Los adaptadores se representan programáticamente por la clase BaseAdapter. Dependiendo de la naturaleza de la lista se elegirá un adaptador prefabricado en el SDK de Android o extenderlos para satisfacer tus necesidades.



Interacción ListView-Adapter

Cuando relacionas un adaptador a una lista, inmediatamente comienza un proceso de comunicación interno para poblarla con una fuente de datos.

Dicha comunicación se basa principalmente en los siguientes métodos del adaptador:

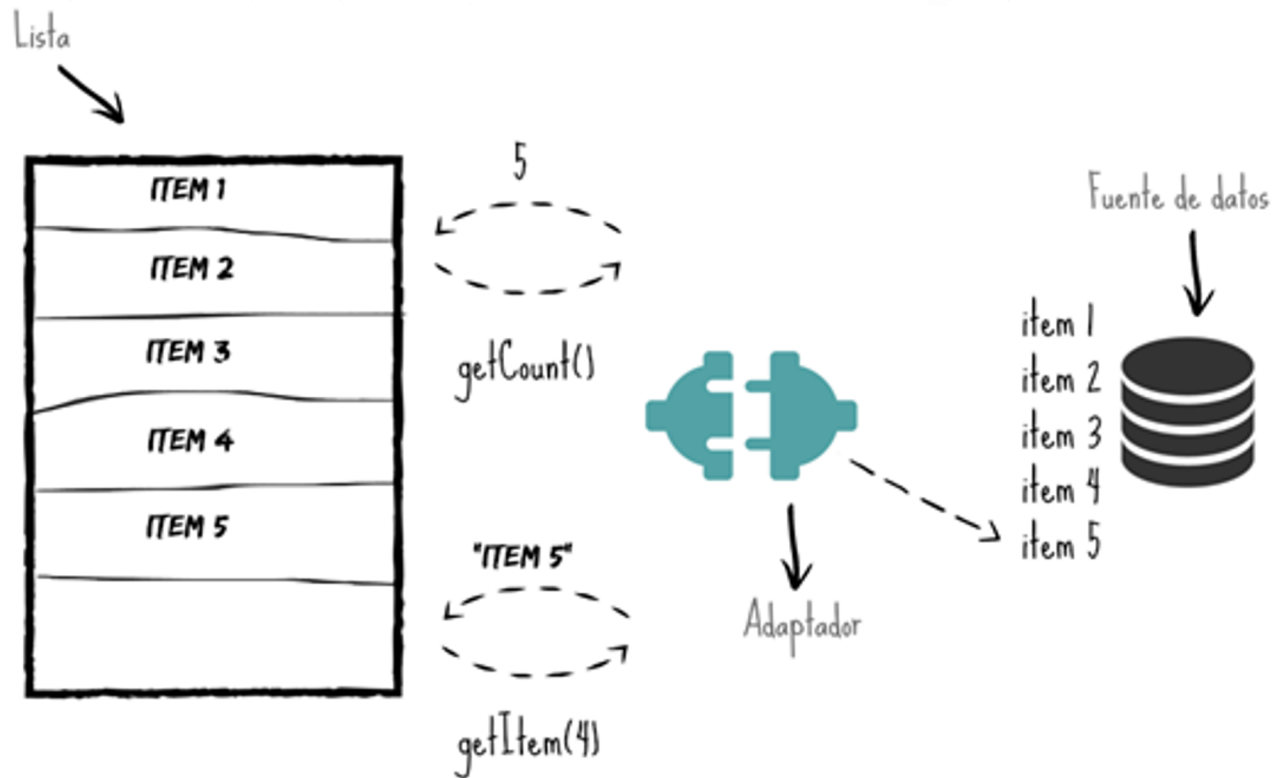
- **getCount():** Retorna en la cantidad de elementos de la fuente de datos. Con este valor la lista ya puede establecer un límite para añadir items.
- **getItem():** Obtiene un elemento de la fuente de datos asignada al adaptador en una posición establecida. Normalmente la fuente de datos es una lista de objetos o un Cursor.
- **getView():** Retorna en el View inflado y ligado a los datos según su posición.

Aunque estos tres métodos no son los únicos que existen para establecer la relación, son los más significativos para entender el concepto de un adaptador.

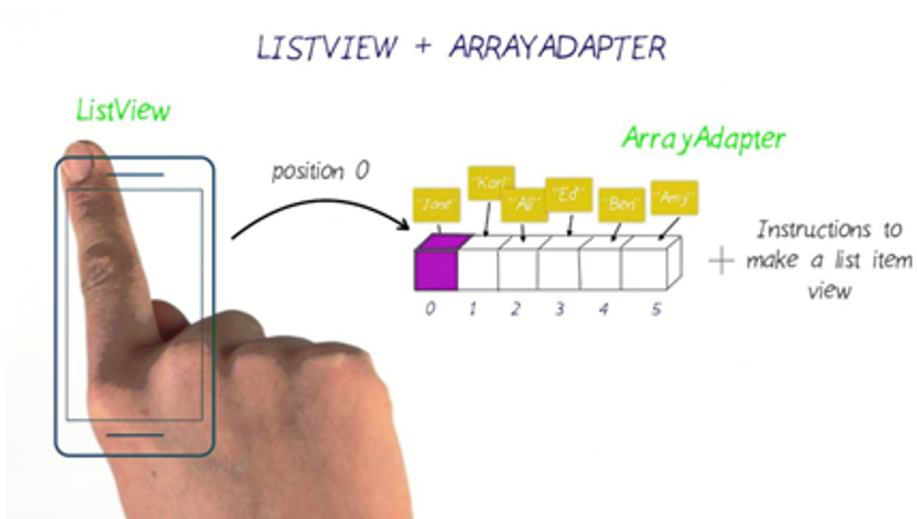
El flujo de interacciones sería el siguiente:

- **ListView:** ¿Cuántos elementos hay para hoy?
- **Adapter:** Existen «getCount()» elementos en la fuente
- **ListView:** ¡Comprendo!... muéstrame el elemento 3
- **Adapter:** Sí claro, el contenido es «getItem(2)»
- **ListView:** Mmm...ya veo... ¿Y cuál sería su View?
- **Adapter:** Al elemento 3 le corresponde «getView(2)»
- **ListView:** Ok, lo mostraré al usuario

Así sería esta relación



La clase ArrayAdapter



Este adaptador infla los ítems con un layout preestablecido de Android, invoca el método **toString()** de cada elemento y lo setea automáticamente en uno o dos **TextViews** según el diseño.

Android provee subclases de adaptadores que nos facilitan la implementación.

ArrayAdapter es uno de los tipos de adaptadores más sencillos y populares.

Debido a que es una **clase genérica**, puedes usar cualquier objeto para poblar la lista.