



Introduction to Android

Digital Systems M, Module 2
Matteo Poggi, Università di Bologna

Introduction to Android

What is Android

Android is an open-source, Linux-based operating system.

It is intended for mobile computing platforms (smartphones, but also smart TVs, wearable, Android auto, etc...) and implements a software stack to develop mobile operating systems

Founded in 2003 and acquired by Google in 2005

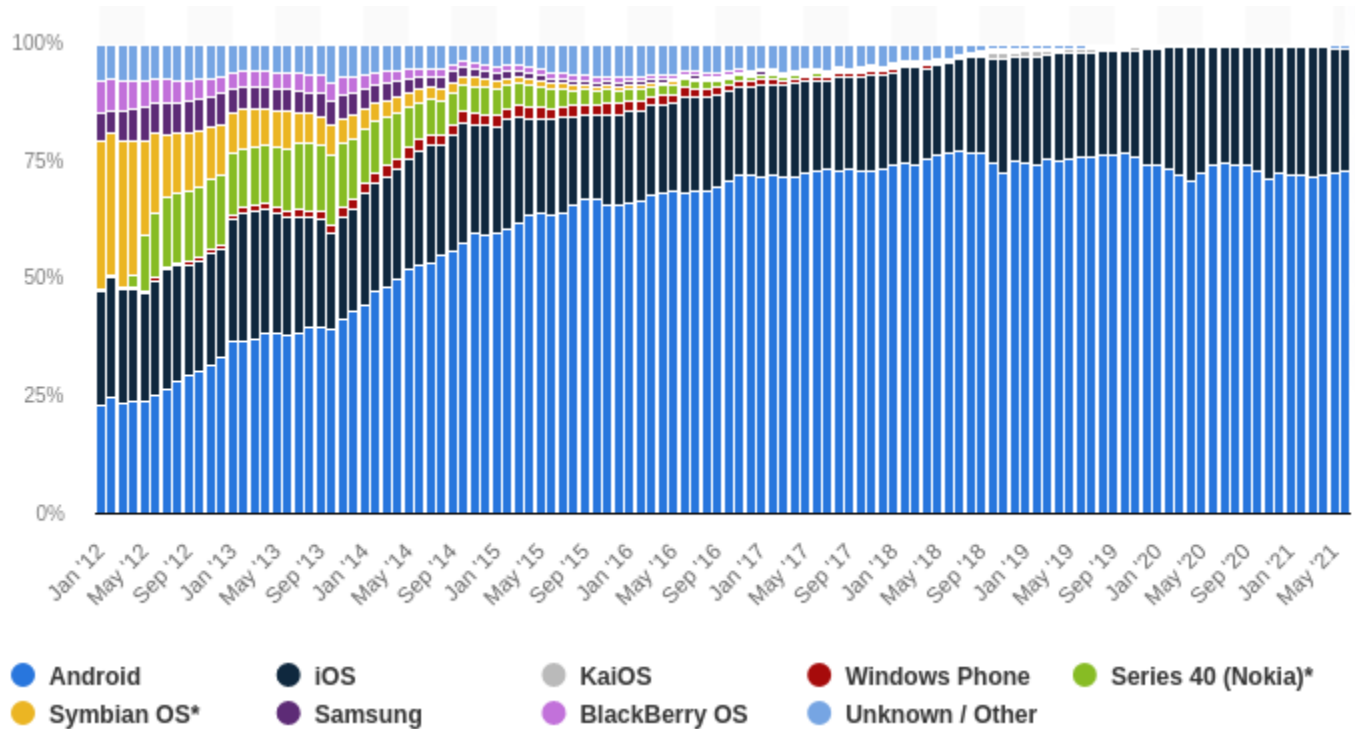
Introduction to Android

Android versions: several versions of Android have been developed and released in the years. Each version is characterized by a set of **supported APIs**



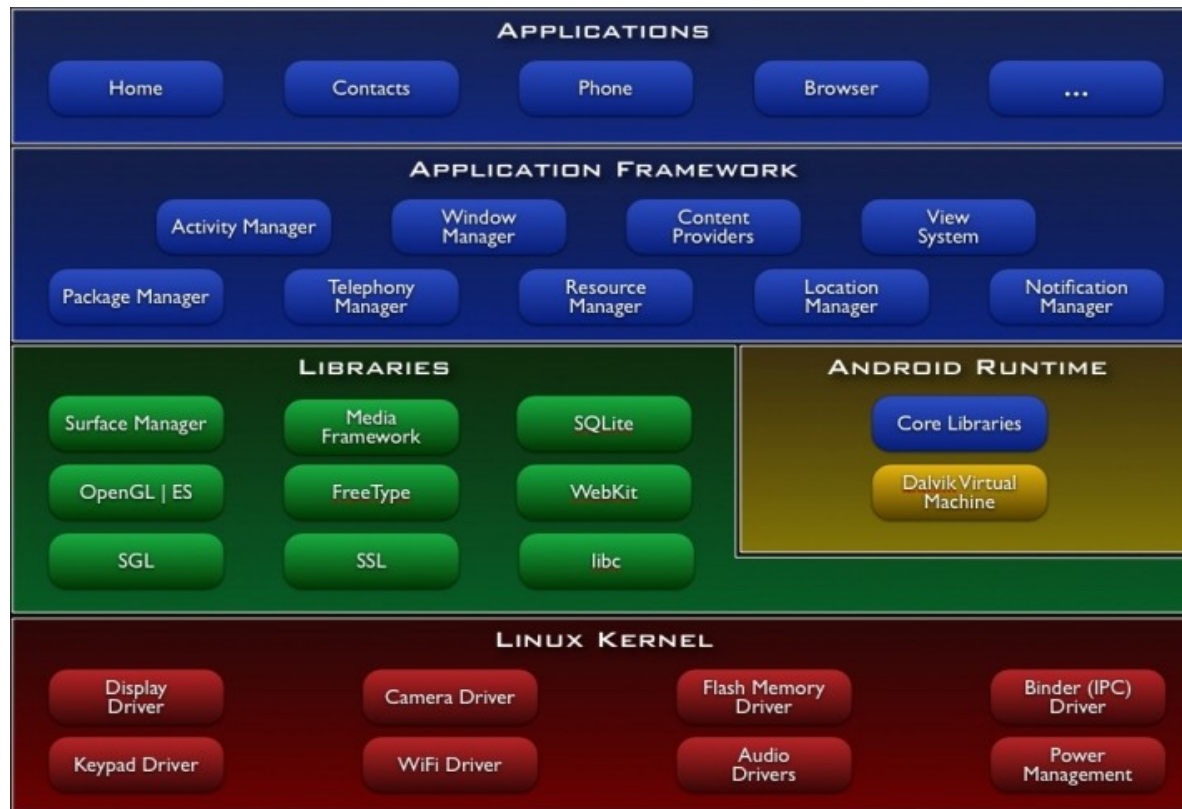
Introduction to Android

Mobile OS distribution worldwide (2012-2021)



Introduction to Android

Android stack architecture



Applications

Highest layer in the Android stacks. It contains core applications shipped with the device itself (for smartphones: SMS manager, email client, contacts, etc.), as well as any application installed by the user.

These applications are traditionally written in **Java**.

Application Framework

It provides classes used to develop Android applications. It handles hardware access by providing generic abstractions to the developer (for instance, Camera API) and manages user interface and application resources

Introduction to Android

Libraries

A set of libraries implementing most of functionalities available in the core Java libraries. Indeed, although Android applications are written in Java, they **do not** run on the Java Virtual Machine.

Android runtime

A set of core libraries providing most of the functions available in Java core libraries. Among them, **Dalvik** is a Java virtual machine specifically designed for Android (in particular, it is written to allow efficient run of multiple Dalvik instances).

Every Android application runs on **its own** Dalvik instance.

Introduction to Android

Android programming would deserve an entire course for itself.

We will mainly focus on:

- 1) Using Android cameras (CameraX APIs)
- 2) Processing images on Android (OpenCV)
- 3) Embedding Python in Android applications (Chaquopy)



Setup

Introduction to Android

The official Integrated Development Environment (IDE) for Android programming is **Android Studio**.

It is based on IntelliJ IDEA and provides several features for fast prototyping

- Gradle-based build system
- fast emulation on a virtual device
- unified environment to develop on many Android devices (smartphones, tablets, smartwatch, etc.)
- C++ and NDK support
- ...

Introduction to Android

Android studio is available for Windows, Mac and Linux at <https://developer.android.com/studio>

You can download it from there and unpack it in *usr/local/* or:

- Install Java SDK

```
sudo apt install openjdk-11-jdk
```

- Add official repository

```
sudo add-apt-repository ppa:maarten-fonville/android-studio
```

```
sudo apt update
```

- Install

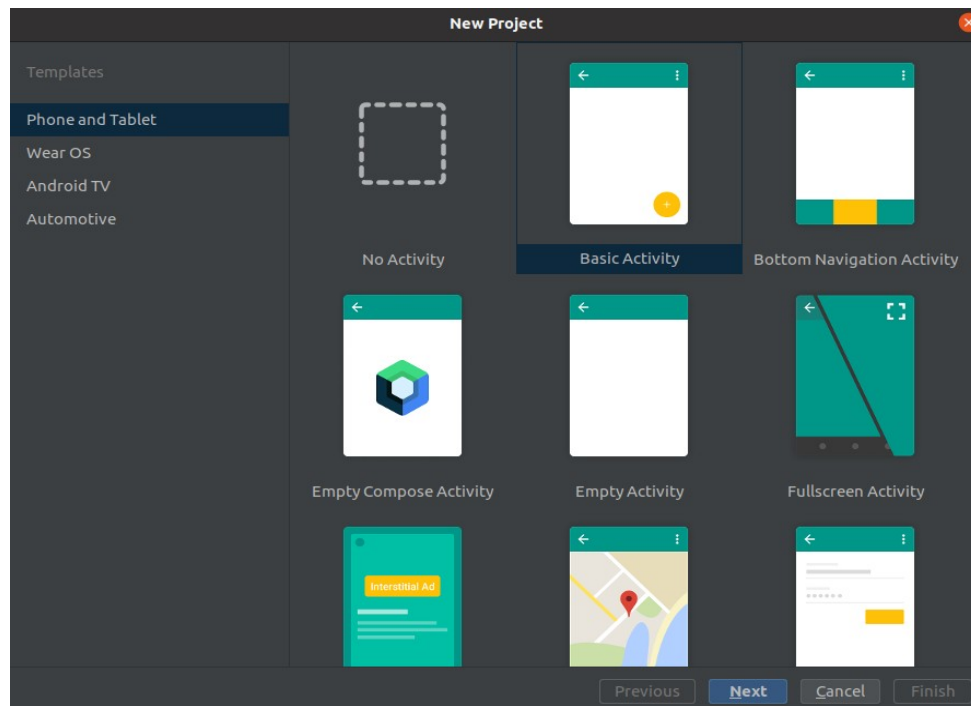
```
sudo apt install android-studio
```



Hello Android

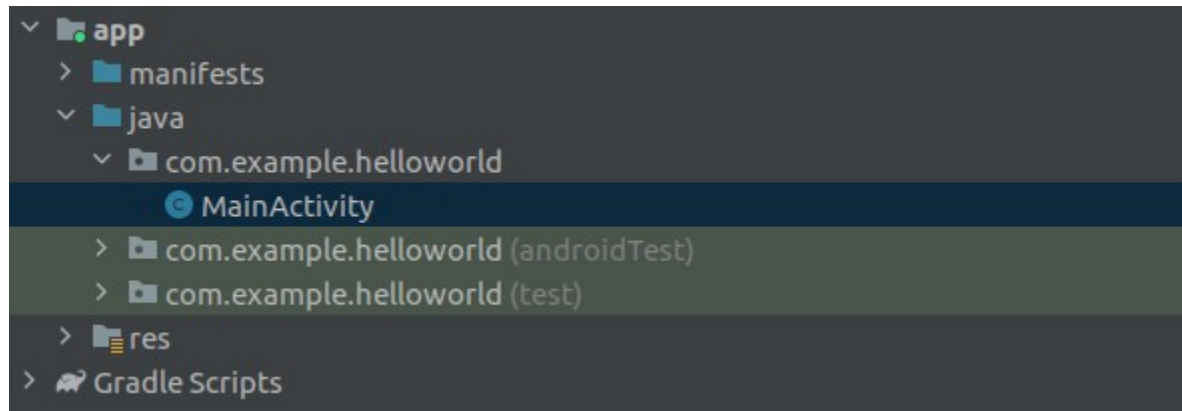
Introduction to Android

Let's start by creating a simple *Helloworld* application



Introduction to Android

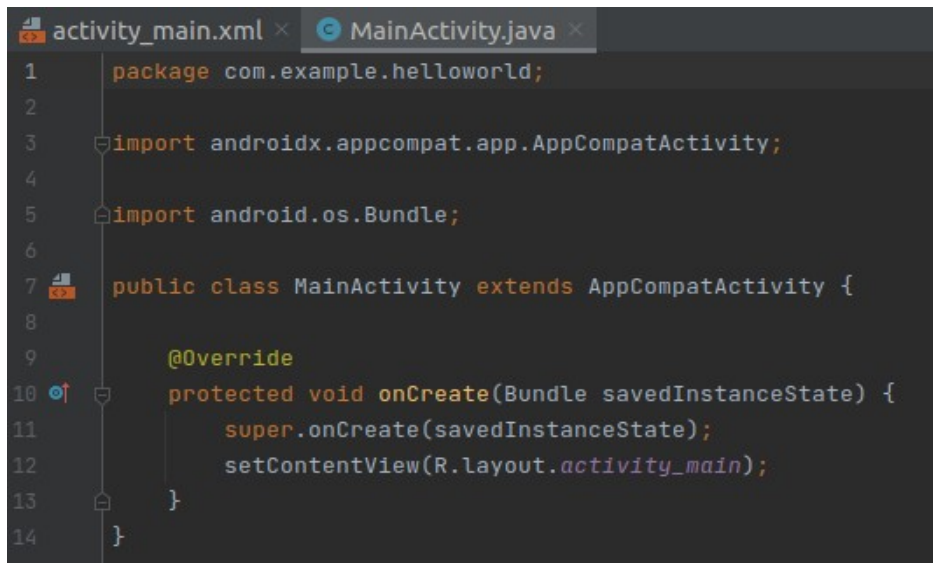
Our newly created project will look organized like this



MainActivity represents the **entry point** in our application

Introduction to Android

MainActivity.java

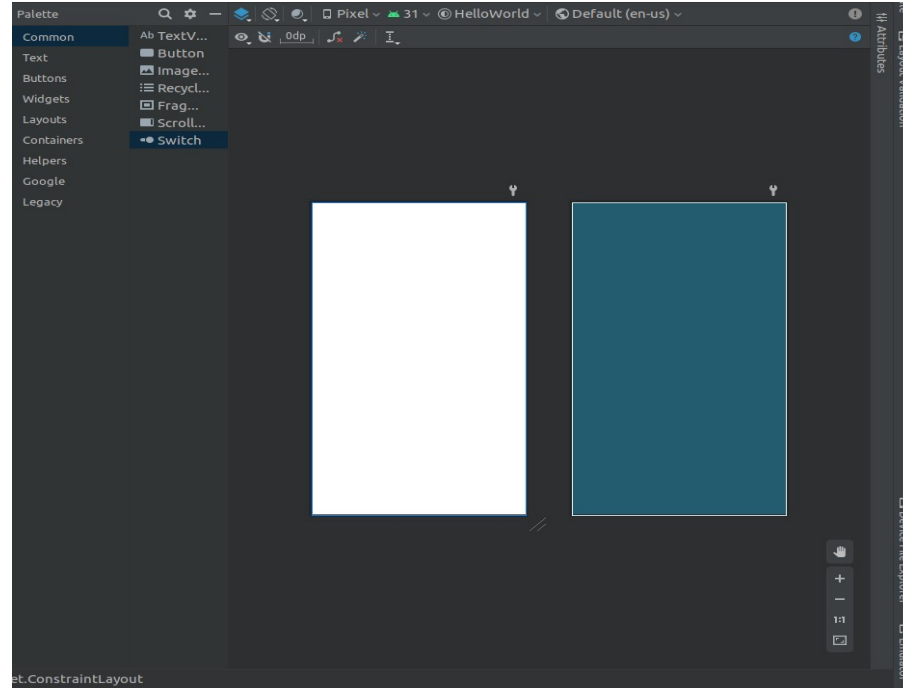
A screenshot of an IDE showing the MainActivity.java file. The code is as follows:

```
1 package com.example.helloworld;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }
```

It plays a role equivalent to the Main class we usually define in Java.
At start, it set the content view to the layout defined in **activity_main.xml**

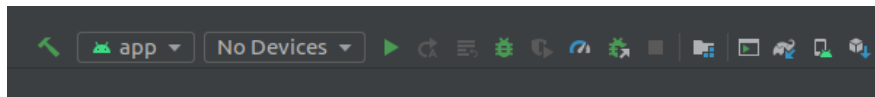
Introduction to Android

Design mode (**design view** vs **blueprint**)

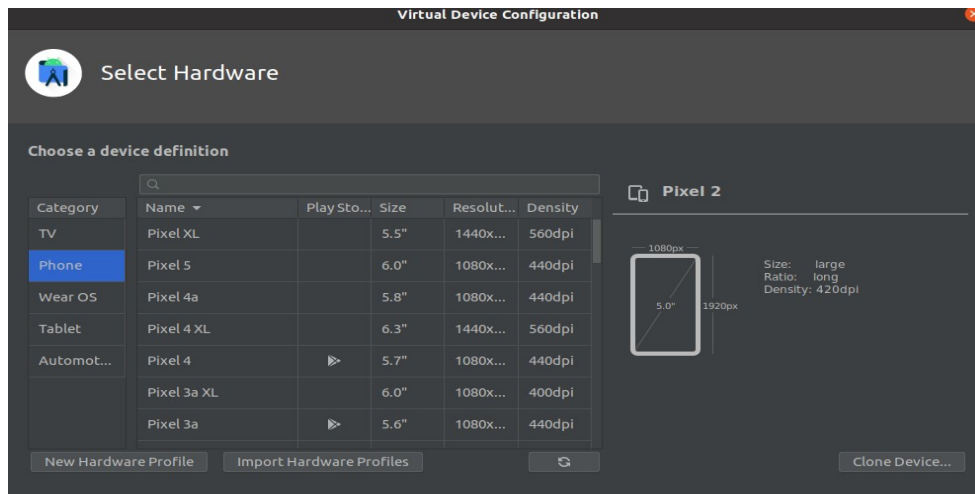


Introduction to Android

Once our application is ready, we can build and run it on Android devices!



We can either connect a real device or create a **virtual one**



Introduction to Android

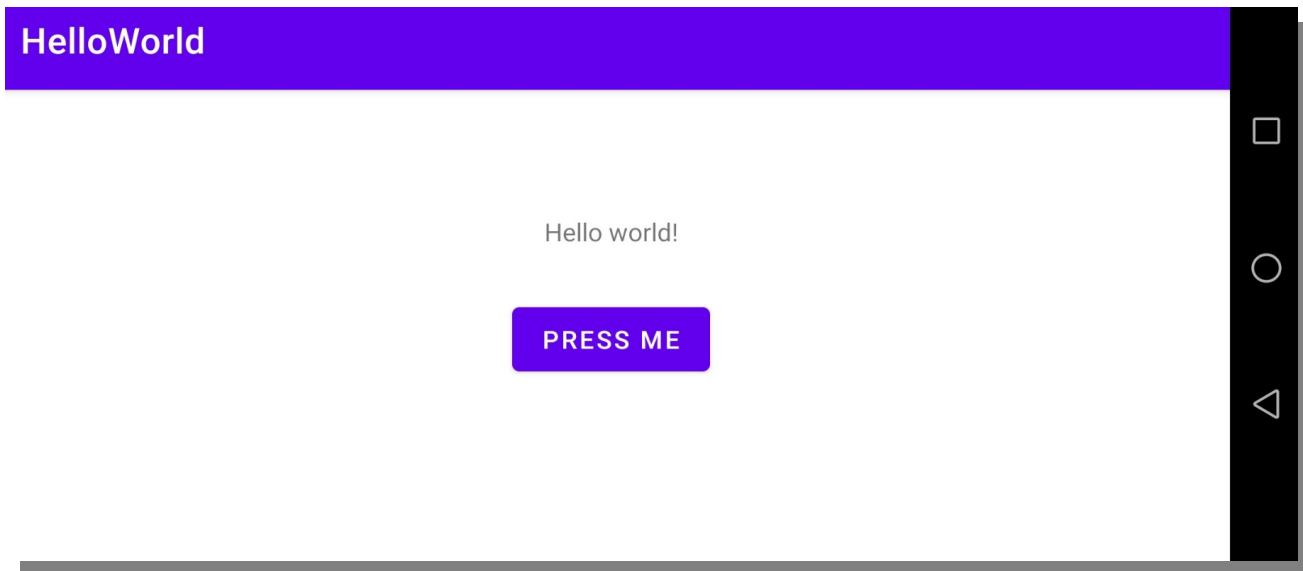
Let's build and run our app on our smartphone. On the device, the application will launch automatically



Introduction to Android

Of course, our application so far is **empty**.

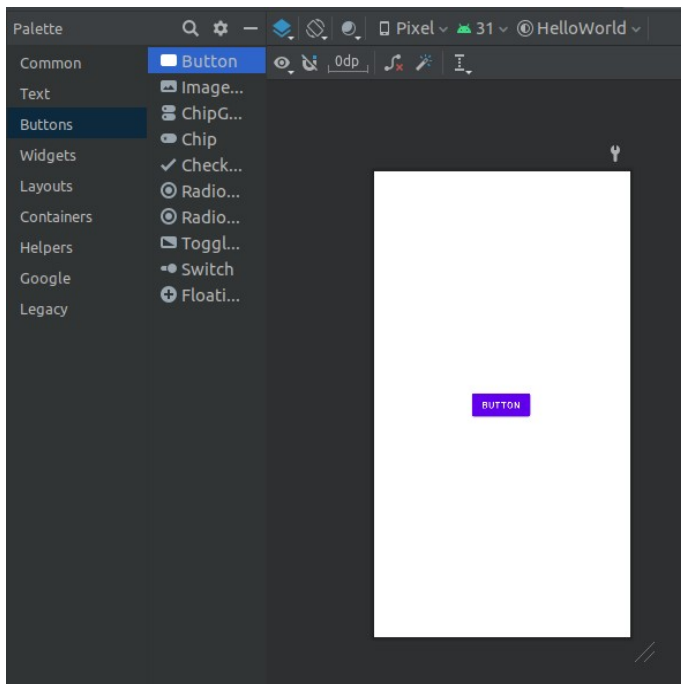
Let's now implement some basic actions, for instance we can add a button that will display "Hello world!" once we press it.



To this aim, we can use **Buttons** and **Text** components

Introduction to Android

From the design view, we can simply drag and drop a button.
In alternative, we can add a Button component in the layout xml file.



```
<Button  
    android:id="@+id/button4"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Button"  
    tools:layout_editor_absoluteX="157dp"  
    tools:layout_editor_absoluteY="343dp" />
```

Introduction to Android

We can edit properties from the layout file. For instance, we can modify the button text by changing the `android:text` attribute.

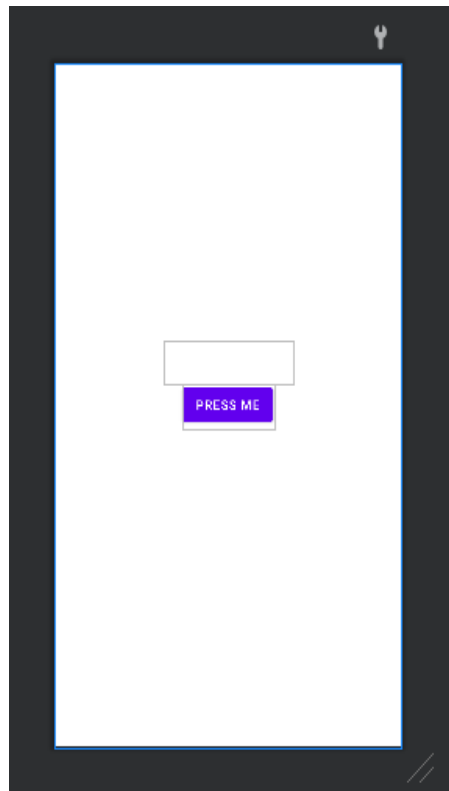
We can move the button and constraint it to the main layout

```
android:text="Press me"  
app:layout_constraintBottom_toBottomOf="parent"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintStart_toStartOf="parent"  
app:layout_constraintTop_toTopOf="parent" />
```

Introduction to Android

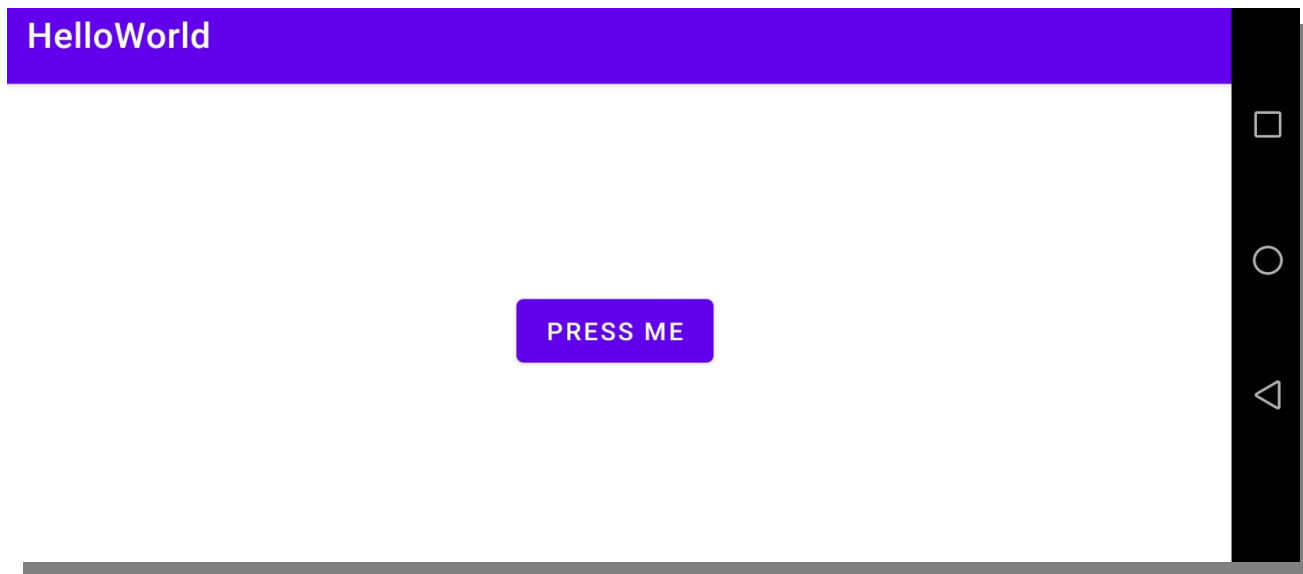
Now, let's add a TextView component and set its text to an empty string.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="154dp"
    android:layout_height="46dp"
    android:text=""
    app:layout_constraintBottom_toTopOf="@+id/button4"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
```



Introduction to Android

By building our application again, we now have our button to press. However, nothing appears when we press it (not a surprise...)



Introduction to Android

We can declare an **handler** linked to our button, that we retrieve by means of its id declared in the xml file:

```
Button button = (Button) findViewById(R.id.button4);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

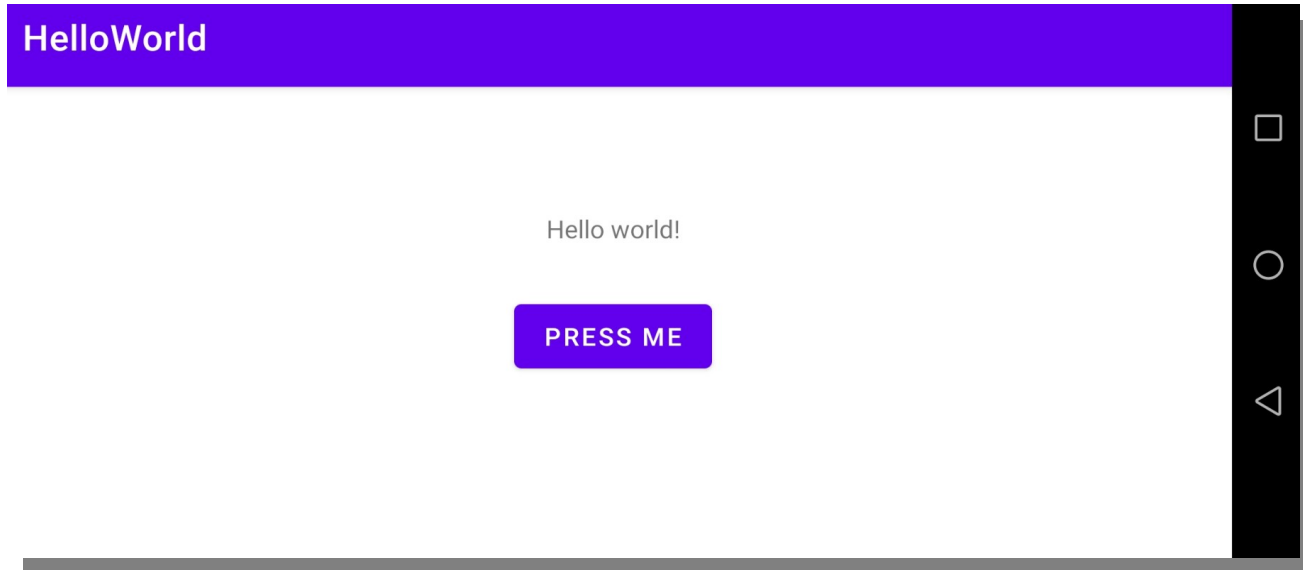
```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{
```

The handler will retrieve the TextView and set its text to “Hello world!”

```
public void onClick(View v) {
    // Do something in response to button click
    TextView t = (TextView) findViewById(R.id.textView);
    t.setText("Hello world!");
}
```


Introduction to Android

Now our button works as intended!



Introduction to Android

We can use **Resources** to make our application more flexible.


Resources are files and static contents that we can use inside our application. They can be both retrieved both in XML or Java files.

Resources are stored in the **res** folder. The main layout file, `activity_main.xml`, is a resource itself!

We will play now with two basic resources, **colors** and **strings**.

Introduction to Android

A set of predefined colors are defined and can be used to draw our GUI.

```
<resources>
  <color name="purple_200">#FFBB86FC</color>
  <color name="purple_500">#FF6200EE</color>
  <color name="purple_700">#FF3700B3</color>
  <color name="teal_200">#FF03DAC5</color>
  <color name="teal_700">#FF018786</color>
  <color name="black">#FF000000</color>
   <color name="white">#FFFFFFFF</color>
</resources>
```

We can define a new color, for instance **red**, by simply adding a new entry

Introduction to Android

A different syntax is used on XML or Java side.

To access strings on XML, we use “@string/name” syntax, with **name** referring to the string name, for instance to set the TextView text color to red.

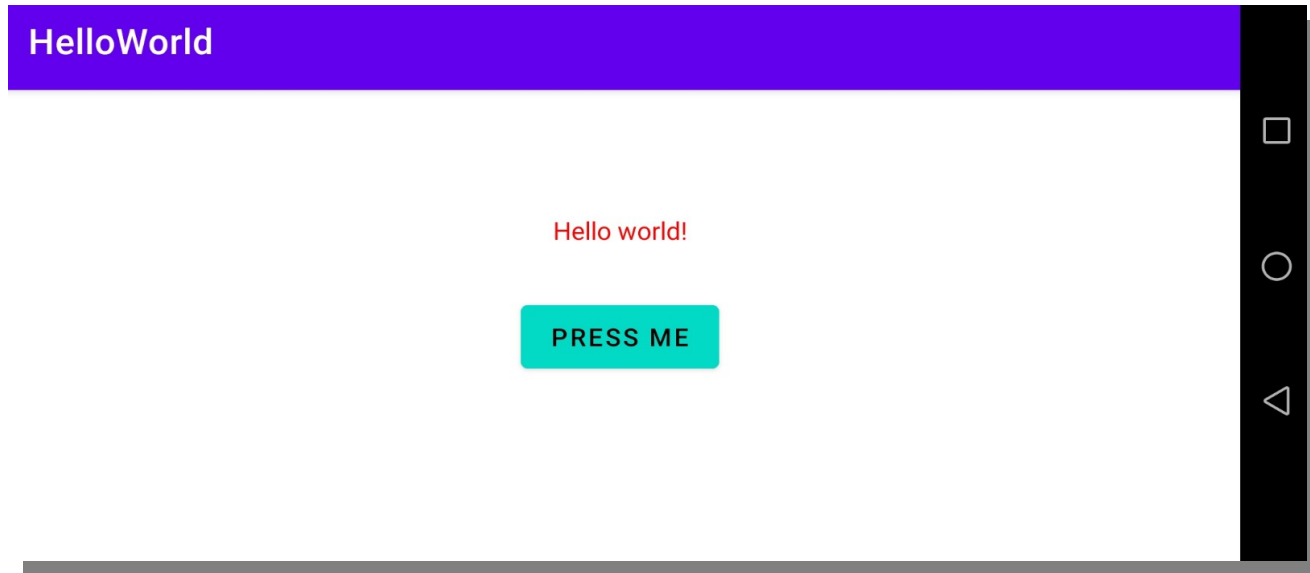
```
android:textColor="@color/red"
```

On Java side, we use R.string.name, for instance to set TextView content after pressing the button.

```
t.setText(R.string.hello_message);
```

Introduction to Android

We can play with colors (and more) and build our application once again!





Hello Android Camera

Introduction to Android

Cameras are nowadays present in **any** smartphone.

Android provides API support for having access to cameras and capture images

Different devices often mean different camera properties (aspect-ratio, orientation, resolution, etc.)

Thus, managing cameras in a consistent way across very different devices is not easy



CameraX APIs

A “simple” interface to access cameras in a consistent manner across multiple devices

Previously, Camera/Camera2 API – much more complex and needed specific functionalities to be implemented according to the device with device-specific code

CameraX is another example of high-level abstraction, taking care for us of any low-level detail!



Use cases

CameraX implements interfaces for specific tasks we want to perform with our camera.

- **Preview** – get an image on our display
- **Image analysis** – run some algorithms (CV or ML) on acquired images
- **Image capture** – save high-res images on storage
- **Video capture** – save video and audio

Introduction to Android

Let's write a simple HelloCamera application.

After creating a blank activity, we first need to add CameraX as a dependency into the gradle build file

```
dependencies {  
  
    implementation 'androidx.appcompat:appcompat:1.3.1'  
    implementation 'com.google.android.material:material:1.4.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'  
    testImplementation 'junit:junit:4.+'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'  
  
    def cameraxVersion = "1.1.0-alpha05"  
    implementation "androidx.camera:camera-core:${cameraxVersion}"  
    implementation "androidx.camera:camera-camera2:${cameraxVersion}"  
    implementation "androidx.camera:camera-lifecycle:${cameraxVersion}"  
}
```

Introduction to Android

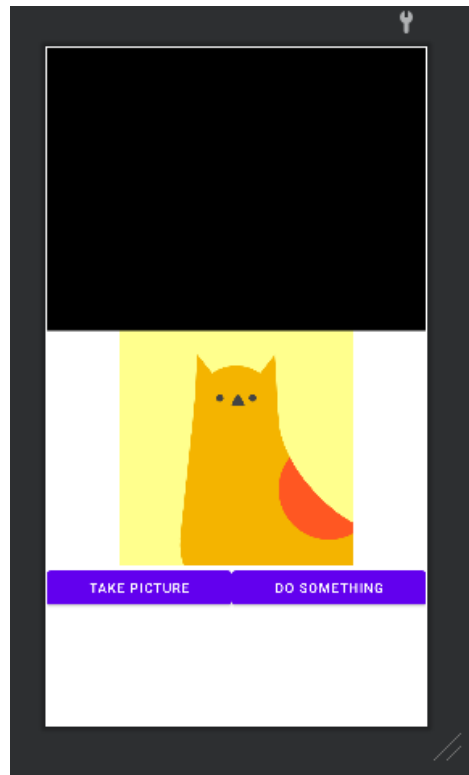
Let's set up the layout.

In this app, we will test **Preview**, **Image Analysis** and **Image Capture** use cases.

To this aim, we add a **previewView** component, an **imageView** and two **buttons**.

The **previewView** is a CameraX custom component, designed to specifically handle the **Preview** use case.

The **imageView** is a standard component which can display images



Introduction to Android

We can use **LinearLayouts** to organize the main layout as a grid of components

We can also define a hierarchy of linear layouts:

- First level: vertical LinearLayout
 - 1st row: previewView
 - 2nd row: imageView
 - 3rd row: horizontal LinearLayout
 - 1st column: capture button
 - 2nd column: analyse button

```
<LinearLayout
    android:layout_width="409dp"
    android:layout_height="729dp"
    android:orientation="vertical"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <androidx.camera.view.PreviewView
        android:id="@+id/previewView"
        android:layout_width="match_parent"
        android:layout_height="279dp"
        android:layout_weight="90" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="252dp"
        android:layout_weight="1"
        tools:srcCompat="@tools:sample/avatars" />

</LinearLayout>
```

Introduction to Android

Before starting with Java code, we need to ensure **permissions**

This is specified in the Android Manifest file

```
<uses-permission android:name="android.permission.CAMERA"/>
```

Starting from Android 11 (API level 30), it needs **explicit user permission**

Introduction to Android

When creating our activity, we can handle it by means of one-time permission request.

```
if (! checkPermission())  
    requestPermission();
```

```
private void requestPermission() {  
  
    ActivityCompat.requestPermissions( activity: this,  
        new String[]{Manifest.permission.CAMERA},  
        PERMISSION_REQUEST_CODE);  
}
```

```
private boolean checkPermission() {  
    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA)  
        != PackageManager.PERMISSION_GRANTED) {  
        // Permission is not granted  
        return false;  
    }  
    return true;  
}
```

```
@Override  
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {  
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);  
    switch (requestCode) {  
        case PERMISSION_REQUEST_CODE:  
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {  
                Toast.makeText(getApplicationContext(), text: "Permission Granted", Toast.LENGTH_SHORT).show();  
                // everything is ok, let's move on  
            } else {  
                Toast.makeText(getApplicationContext(), text: "Permission Denied", Toast.LENGTH_SHORT).show();  
                // Handle this.... asking again or shutting down  
            }  
            break;  
        }  
    }  
}
```

Introduction to Android

Now, we are ready to code. Let's start from our Main Activity and extend it to deal with the specific use cases we want to implement

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener, ImageAnalysis.Analyzer {  
    private ListenableFuture<ProcessCameraProvider> provider;  
  
    private Button picture_bt, analysis_bt;  
    private PreviewView pview;  
    private ImageView imview;  
    private ImageCapture imageCapt;  
    private ImageAnalysis imageAn;  
    private boolean analysis_on;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        picture_bt = findViewById(R.id.picture_bt);  
        analysis_bt = findViewById(R.id.analysis_bt);  
        pview = findViewById(R.id.previewView);  
        imview = findViewById(R.id.imageView);  
  
        picture_bt.setOnClickListener(this);  
        analysis_bt.setOnClickListener(this);  
        this.analysis_on = false;  
    }  
}
```

Introduction to Android

- We implement, within the Main Activity, **onClickListener** and **Analyser** abstract interfaces. The former to handle buttons callbacks (nothing new here), the latter to deal with the **Image Analysis** use cases, by implementing a proper call back as well (see later)

```
implements View.OnClickListener, ImageAnalysis.Analyzer
```

- We retrieve the main components from the layout we designed (nothing new here)

```
picture_bt = findViewById(R.id.picture_bt);  
analysis_bt = findViewById(R.id.analysis_bt);  
pview = findViewById(R.id.previewView);  
imview = findViewById(R.id.imageView);  
  
picture_bt.setOnClickListener(this);  
analysis_bt.setOnClickListener(this);  
this.analysis_on = false;
```


Introduction to Android

The **ProcessCameraProvider** is a singleton, used to bind the camera lifecycle to the lifecycle owner in the application process.

This entity is implemented as an **Executor**. Specifically, we bind the **ProcessCameraProvider** to the main **Executor** in the application.

```
provider = ProcessCameraProvider.getInstance( context: this);
provider.addListener( () ->
{
    try{
        ProcessCameraProvider cameraProvider = provider.get();
        startCamera(cameraProvider);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}, getExecutor());
```

```
private Executor getExecutor() { return ContextCompat.getMainExecutor( context: this); }
```

Introduction to Android

Let's see how to start camera acquisition.

First, unbind all the use cases from the lifecycle and remove them.

Then, we get a **CameraSelector**, allowing to select the camera we will use in our application (front or back facing)

```
private void startCamera(ProcessCameraProvider cameraProvider) {  
    cameraProvider.unbindAll();  
    CameraSelector camSelector = new CameraSelector.Builder().requireLensFacing(CameraSelector.LENS_FACING_BACK).build();  
  
    Preview preview = new Preview.Builder().build();  
    preview.setSurfaceProvider(pview.getSurfaceProvider());  
  
    ImageCapt = new ImageCapture.Builder().setCaptureMode(ImageCapture.CAPTURE_MODE_MINIMIZE_LATENCY).build();  
    imageAn = new ImageAnalysis.Builder().setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST).build();  
    imageAn.setAnalyzer(getExecutor(), analyzer: this);  
  
    cameraProvider.bindToLifecycle((LifecycleOwner) this, camSelector, preview, imageCapt, imageAn);  
}
```

Introduction to Android

Then, we initialize our 3 use cases: Preview, Capture and Analysis.

We bind our **Preview** case to our **previewView** – images will flow on it.

We set the **Analyzer** of our **Analysis** case to the Main Activity **itself**.

Finally, we bind CameraSelector and cases to the Activity lifecycle.

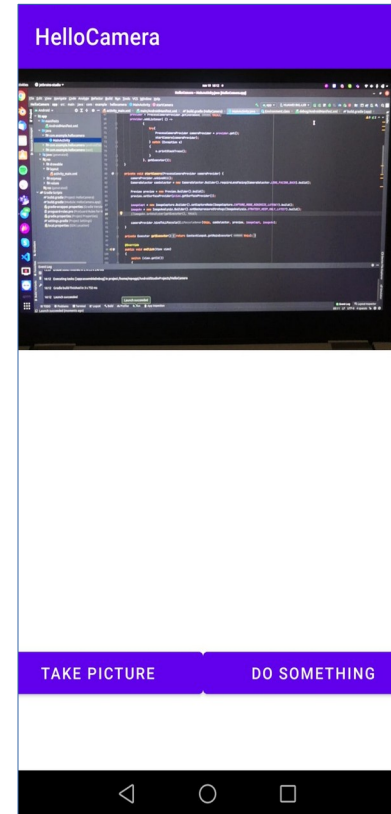
```
private void startCamera(ProcessCameraProvider cameraProvider) {  
    cameraProvider.unbindAll();  
    CameraSelector camSelector = new CameraSelector.Builder().requireLensFacing(CameraSelector.LENS_FACING_BACK).build();  
  
    Preview preview = new Preview.Builder().build();  
    preview.setSurfaceProvider(pview.getSurfaceProvider());  
  
    ImageCapt = new ImageCapture.Builder().setCaptureMode(ImageCapture.CAPTURE_MODE_MINIMIZE_LATENCY).build();  
    imageAn = new ImageAnalysis.Builder().setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST).build();  
    imageAn.setAnalyzer(getExecutor(), analyzer: this);  
  
    cameraProvider.bindToLifecycle((LifecycleOwner) this, camSelector, preview, imageCapt, imageAn);  
}
```

Introduction to Android

Let's build and run our application.

Once started, we can see on top, in the **previewView**, a video stream collected live by the smartphone camera.

Let's now implement the callbacks to capture images and process them



Introduction to Android

In the **onClick** callback, we discriminate between the two buttons in our view and implement different behaviors accordingly.

The capture button will call a **capturePhoto** function (see later)

For the analysis button, we implement a switching mechanism to turn on/off the analysis process, implemented inside the **analyse** callback (see later)

```
@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.picture_bt:
            capturePhoto();
            break;

        case R.id.analysis_bt:
            this.analysis_on = !this.analysis_on;
            break;
    }
}
```

Introduction to Android

We first prepare the filename to save our picture, then we call the **takePicture** function from the ImageCapture use case, defining an **OnImageCaptureCallback**

```
public void capturePhoto() {  
    //Es. SISDIG_2021127_189230.jpg  
    String pictureName = "SISDIG_" + new SimpleDateFormat(pattern: "yyyyMMdd_HHmmss").format(new Date()) + ".jpeg";  
    imageCapt.takePicture(  
        getExecutor(),  
        new ImageCapture.OnImageCapturedCallback() {
```

Inside the callback, we implement the saving logic by overriding the **OnCaptureSuccess** function, taking as input an **ImageProxy** object, a wrapper for an Android Image

Introduction to Android

The **MediaStore** class allows applications to access media.

We add the picture to the MediaStore itself, to make it accessible to other apps

Then, we save it using an **OutputStream**

Credits: Cristian Davide Conte

```
new ImageCapture.OnImageCapturedCallback() {
    @Override
    public void onCaptureSuccess(ImageProxy image) {
        //Create the picture's metadata
        ContentValues newPictureDetails = new ContentValues();
        newPictureDetails.put(MediaStore.Images.Media._ID, pictureName);
        newPictureDetails.put(MediaStore.Images.Media.ORIENTATION, String.valueOf(-image.getImageInfo().getRotationDegrees()));
        newPictureDetails.put(MediaStore.Images.Media.DISPLAY_NAME, pictureName);
        newPictureDetails.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
        newPictureDetails.put(MediaStore.Images.Media.WIDTH, image.getWidth());
        newPictureDetails.put(MediaStore.Images.Media.HEIGHT, image.getHeight());
        newPictureDetails.put(MediaStore.Images.Media.RELATIVE_PATH, Environment.DIRECTORY_DCIM + "/" + "SistemiDigitaliM");
        OutputStream stream = null;
        try {
            //Add picture to MediaStore in order to make it accessible to other apps
            //The result of the insert is the handle to the picture inside the MediaStore
            Uri picturePublicUri = getApplicationContext().getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, newPictureDetails);
            stream = getApplicationContext().getContentResolver().openOutputStream(picturePublicUri);
            Bitmap bitmapImage = convertImageProxyToBitmap(image);
            if (bitmapImage.compress(Bitmap.CompressFormat.JPEG, 100, stream)) { //Save the image in the gallery
                //Error
            }

            image.close();
            stream.close();

            Toast.makeText(getApplicationContext(), text: "Picture Taken", Toast.LENGTH_SHORT).show();
        } catch (Exception exception) {
            exception.printStackTrace();
            Toast.makeText(getApplicationContext(), text: "Error saving photo: " + exception.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
}
```

Introduction to Android

However, **Bitmap** objects are a standard way to handle images. Most Android libraries (OpenCV has an Android version too!) work on Bitmap objects.

We need to convert ImageProxy objects to Bitmaps to pass them to the **OnCaptureSuccess** function

```
private Bitmap convertImageProxyToBitmap(ImageProxy image) {  
    ByteBuffer byteBuffer = image.getPlanes()[0].getBuffer();  
    byteBuffer.rewind();  
    byte[] bytes = new byte[byteBuffer.capacity()];  
    byteBuffer.get(bytes);  
    BitmapFactory.decodeByteArray(bytes, offset: 0, bytes.length);  
    Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, offset: 0, bytes.length);  
    Matrix matrix = new Matrix();  
    matrix.postRotate(image.getImageInfo().getRotationDegrees());  
    bitmap = Bitmap.createBitmap(bitmap, x: 0, y: 0, image.getWidth(), image.getHeight(), matrix, filter: false);  
    return bitmap;  
}
```


Introduction to Android

Let's have a look to the analyse callback. It will process the last image acquired as an ImageProxy

In alternative, we can also analyze the latest image shown by the **PreviewView**

According to the documentation, it is good practice to bind the use case to the Analyzer **only once** to avoid overhead

```
@Override
public void analyze(@NonNull ImageProxy image) {
    if(this.analysis_on)
    {
        Bitmap conv = convertImageProxyToBitmap(image);
        // Do something here!!!
        conv = toGrayscale(conv);
        this.imview.setImageBitmap( conv );
    }
    image.close();
}
```

```
@Override
public void analyze(@NonNull ImageProxy image) {
    if(this.analysis_on)
    {
        Bitmap conv = pview.getBitmap();
        // Do something here!!!
        conv = toGrayscale(conv);
        this.imview.setImageBitmap( conv );
    }
    image.close();
}
```

Introduction to Android

In this example, our analysis concerns converting RGB images to grayscale format.

Canvas and **Paint** objects allow for drawing content inside a bitmap.

ColorMatrixColorFilter implements color filtering (setting saturation to 0 convert the images to grayscale)

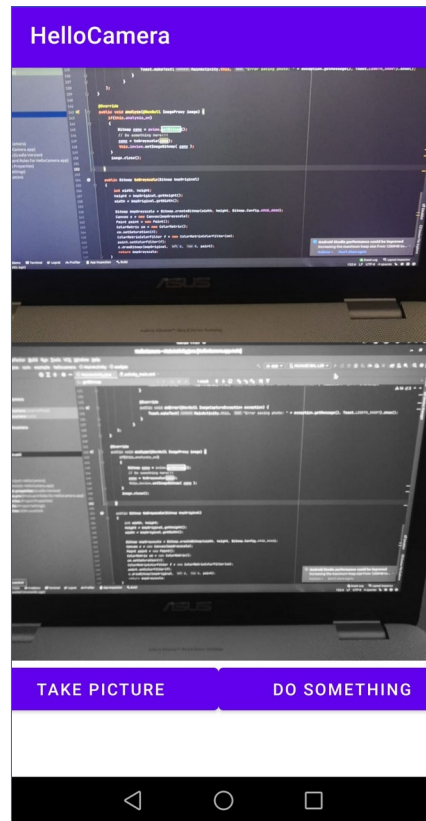
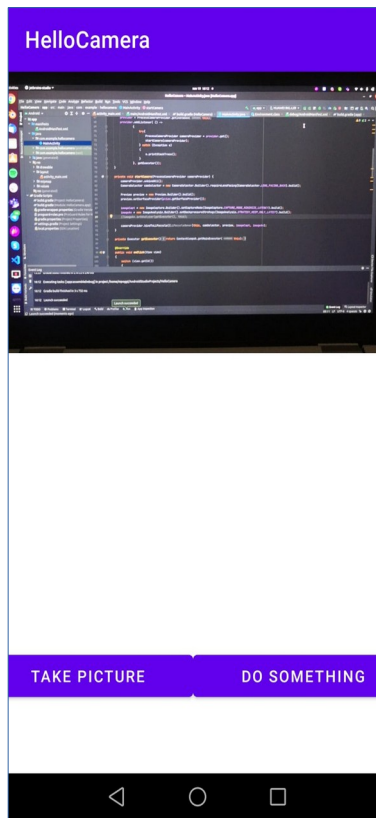
```
public Bitmap toGrayscale(Bitmap bmpOriginal)
{
    int width, height;
    height = bmpOriginal.getHeight();
    width = bmpOriginal.getWidth();

    Bitmap bmpGrayscale = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
    Canvas c = new Canvas(bmpGrayscale);
    Paint paint = new Paint();
    ColorMatrix cm = new ColorMatrix();
    cm.setSaturation(0);
    ColorMatrixColorFilter f = new ColorMatrixColorFilter(cm);
    paint.setColorFilter(f);
    c.drawBitmap(bmpOriginal, left: 0, top: 0, paint);
    return bmpGrayscale;
}
```

Introduction to Android

Let's build again our application.

After pressing “**do something**”,
the **imageView** will display
the grayscale converted
image we see in the
previewView





Hello OpenCV

Introduction to Android

Several modules are already implemented and ready to be used on Android devices. Among them, we have an OpenCV version for Android!

It provides implementation for most algorithms available in standard OpenCV distributions.

To use it in our application, we can download it from the official website, add it to our project as a module (File → New → import module, and select opencvfolder/sdk/java) and add the dependency in our manifest file.

```
implementation 'com.quickbirdstudios:opencv:3.4.1'
```

Introduction to Android

To embed OpenCV functionalities in our application, we need to import **OpenCVLoader**

When creating our main activity, we need to initialize the OpenCV module before running any OpenCV function

```
import org.opencv.android.OpenCVLoader;  
import org.opencv.android.Utils;  
import org.opencv.core.Mat;  
import org.opencv.imgproc.Imgproc;
```

```
static {  
    if (!OpenCVLoader.initDebug())  
        Log.d( tag: "ERROR", msg: "Unable to load OpenCV");  
    else  
        Log.d( tag: "SUCCESS", msg: "OpenCV loaded");  
}
```

Now, let's implement our color conversion using OpenCV!

Introduction to Android

Most OpenCV functions work on **Mat** objects (as in C++)

Mat are abstractions for matrix objects (images are matrices....)

Utils provide several utility functions, such as conversions between Bitmap and Map objects

Imgproc contains most OpenCV Algorithms, for instance **cvtColor**

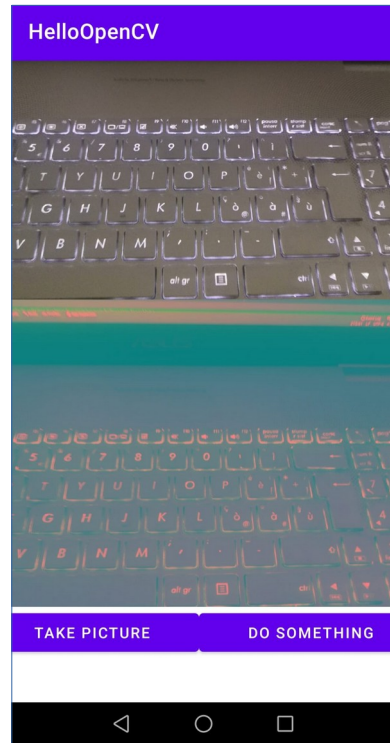
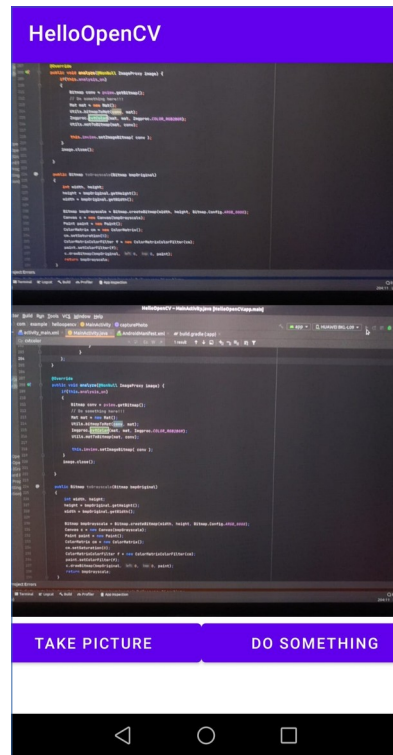
```
@Override
public void analyze(@NonNull ImageProxy image) {
    if(this.analysis_on)
    {
        Bitmap conv = pview.getBitmap();
        // Do something here!!!
        Mat mat = new Mat();
        Utils.bitmapToMat(conv, mat);
        Imgproc.cvtColor(mat, mat, Imgproc.COLOR_RGB2GRAY);
        Utils.matToBitmap(mat, conv);

        this.imview.setImageBitmap( conv );
    }
    image.close();
}
```


Introduction to Android

cvtColor works exactly as the python counterpart we have seen some weeks ago

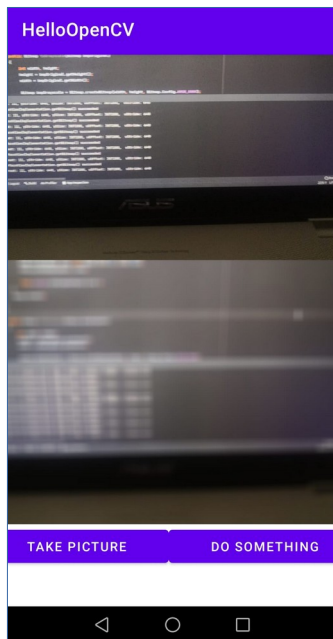
Then, we can easily convert our image to grayscale, as well as to different encodings such as BGR or YUV



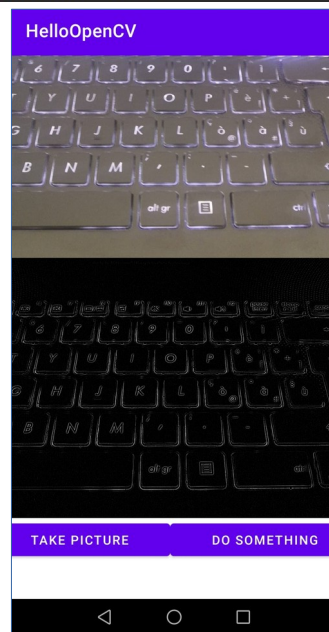
Introduction to Android

Do you remember filters? We can easily filter images as we did in python!

```
Utils.bitmapToMat(conv, mat);  
Imgproc.blur(mat, mat, new Size( width: 31, height: 31));  
Utils.matToBitmap(mat, conv);
```



```
Utils.bitmapToMat(conv, mat);  
Imgproc.cvtColor(mat, mat, Imgproc.COLOR_RGB2GRAY);  
Imgproc.Laplacian(mat, mat, ddepth: 8, ksize: 3, scale: 1, delta: 0);  
Utils.matToBitmap(mat, conv);
```



Exercise: implement an Android application with a set of four filters: average, median, Laplacian and Gaussian.

The user should be able to select which filter to apply by pressing one of four buttons.



Hello Chaquopy

We have learned to appreciate Python during the whole course...

... so let's use Python on Android as well!

Chaquopy is a Python SDK though for Android systems, with

- full integration with Android Studio and Gradle
- APIs for calling Python code on Java/Kotlin side (or vice-versa)
- Support to most popular Python packages (opencv, matplotlib, etc.)

Introduction to Android

Setting up Chaquopy is easy, we only need to edit Gradle build files

On the top file, we add the remote repository and the packages to be installed and Gradle will handle everything (check versions compatibility...)

On the app build file, we specify the native compilers, the python path (in our developer device) and optional pip packages required by our python code

```
buildscript {  
    repositories {  
        maven { url "https://chaquo.com/maven" }  
    }  
    dependencies {  
        classpath "com.android.tools.build:gradle:7.0.2"  
        classpath "com.chaquo.python:gradle:10.0.1"  
    }  
}
```

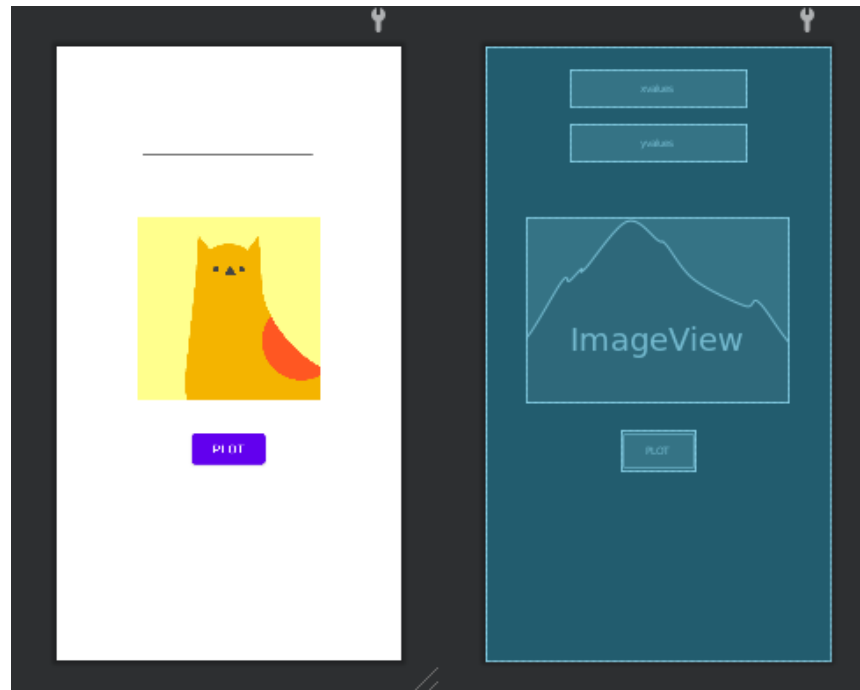
```
ndk {  
    abiFilters "armeabi-v7a", "arm64-v8a", "x86", "x86_64"  
}  
  
python {  
    buildPython "/usr/bin/python"  
    pip {  
        install "numpy"  
        install "matplotlib"  
    }  
    pyc {  
        src false  
    }  
}
```

Introduction to Android

We are now going to design an Android app to draw a plot given a set of comma-separated x values and y values from two **EditText** views.

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {  
  
    Button plot_bt;  
    ImageView imview;  
    EditText xvalues, yvalues;  
    String xtext, ytext;  
    Python py;  
}
```

We create buttons and views,
together with a **Python** object.



Introduction to Android

During creation, we get the reference to any element and we start the **Python engine**

Then, we get an instance of the Python and will use it to run our Python code

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

    plot_bt = (Button)findViewById(R.id.button);
    imview = (ImageView)findViewById(R.id.imageView);

    xvalues = (EditText)findViewById(R.id.xvalues);
    yvalues = (EditText)findViewById(R.id.yvalues);

    if(!Python.isStarted())
        Python.start(new AndroidPlatform(context: this));

    py = Python.getInstance();

    plot_bt.setOnClickListener(this);
}
```

Introduction to Android

Let's take a look to the Python side

We define a function processing two strings, x and y, to draw our plot

We process the strings to extract our x and y values, and plot them by using **matplotlib** functions

Finally, we return a **byte array**

```
import matplotlib.pyplot as plt
import numpy as np
import io

def main(x,y):
    x_data = [int(i) for i in x.split(',')]
    y_data = [int(j) for j in y.split(',')]

    fig, ax = plt.subplots()
    ax.plot(x_data, y_data)

    f = io.BytesIO()
    plt.savefig(f, format="png")

    return f.getvalue()
```


Introduction to Android

Let's put pieces together. On Java side, we implement the button callback, extracting text strings from the two EditText views

We load our python script in a **PyObject** – an interface to Python data structures. Then, we run the **main** function from our script through the **callAttr** function, passing also the two strings as inputs

This function returns a PyObject. We convert it to a byte array and use this latter to draw a **Bitmap** and, finally, set it into the **imageView**.

```
@Override
public void onClick(View v) {
    xtext = xvalues.getText().toString();
    ytext = yvalues.getText().toString();

    PyObject pyo = py.getModule( name: "plotter");
    PyObject obj = pyo.callAttr( key: "main", xtext, ytext);

    byte data[] = obj.toJava(byte[].class);
    Bitmap plot = BitmapFactory.decodeByteArray(data, offset: 0, data.length);
    imview.setImageBitmap(plot);
}
```

Introduction to Android

Here the final result!

Thanks to Chaquopy, we can embed several Python libraries in our Android applications

Scikit-learn, OpenCV itself, ...

..., Tensorflow, ...

