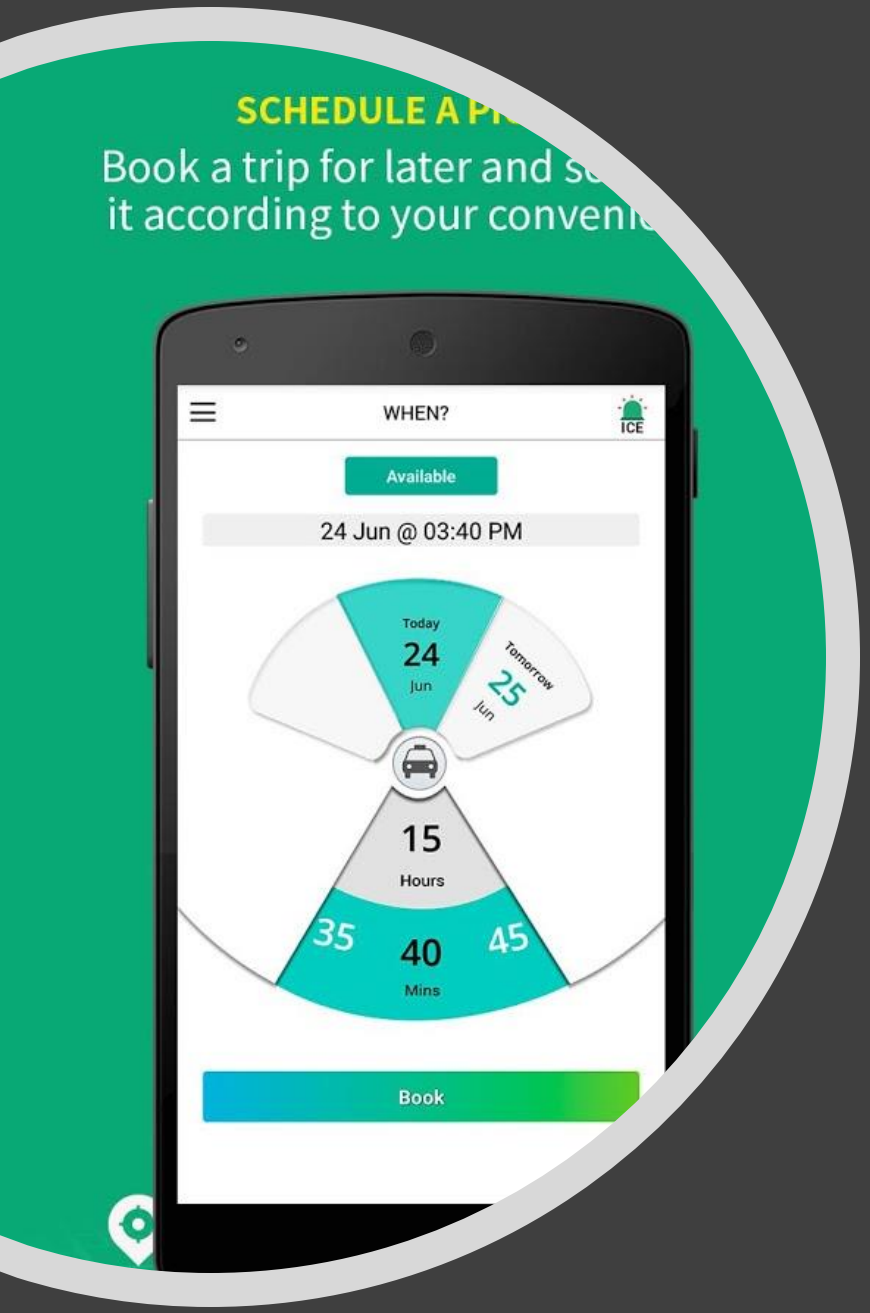
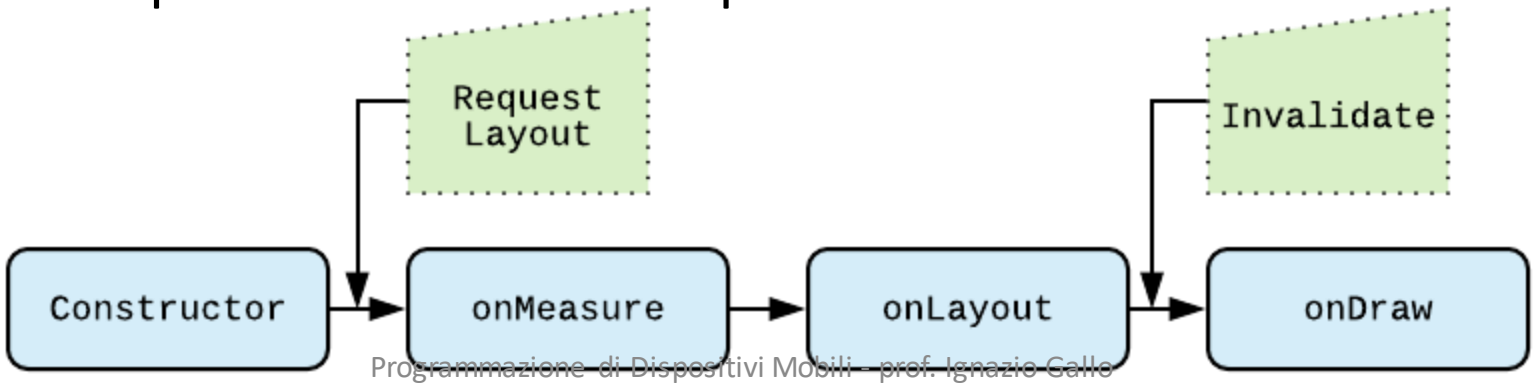


Custom Views

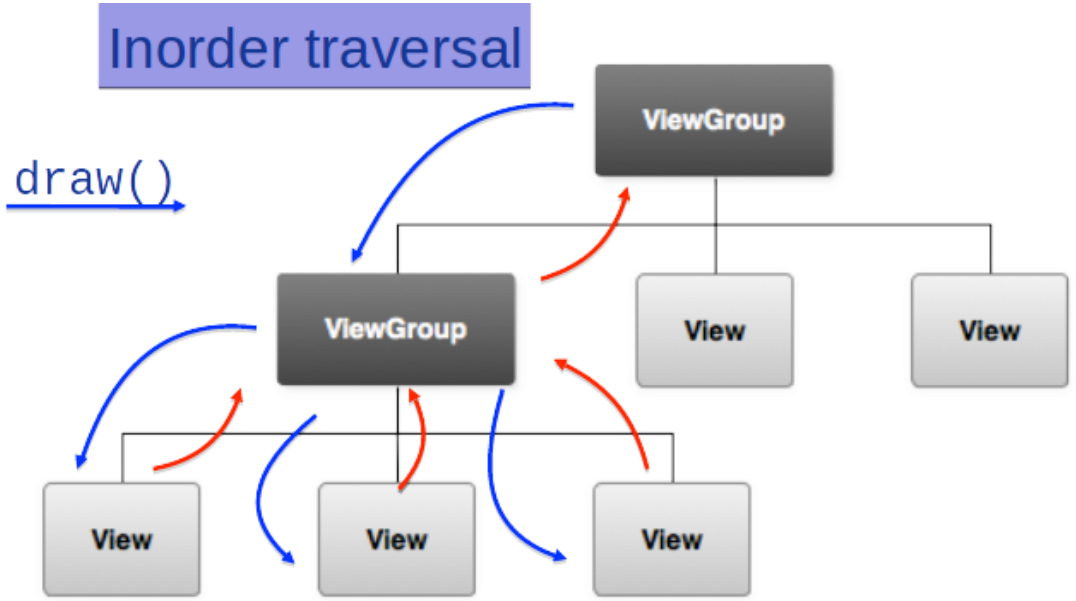
How Android draws Views

- When an Activity receives focus, it will be requested to draw its layout.
- Drawing the layout is a two pass process:
  - **measure**: is a top-down traversal of the View tree. Each View pushes dimension specifications down the tree during the recursion. At the end of the measure pass, every View has stored its measurements.
  - **layout**: is a top-down traversal of the View tree.
    Each parent is responsible for positioning and drawing all of its children using the sizes computed in the measure pass.



How Android draws Views

- Drawing is handled by walking the tree and rendering each View that intersects the invalid region.
  - the invalid region is the one which need to be drawn.
- Each ViewGroup is responsible for requesting each of its children to be drawn (with the draw() method).
- Each View is responsible for drawing itself.



- Parents are drawn before (i.e., behind) their children.
- Siblings are drawn in the order they appear in the tree.

Custom Components

- You can define custom components which can be used either from code or as XML elements.
- To create a fully customized component:
  - create an extension of a View;
    - public View(Context context)
    - public View(Context context, AttributeSet attrs)
  - Initialize the custom view with drawing and painting values.
  - Override `onDraw()` to draw the view.
  - Use listeners to provide the custom view's behavior.
  - Add the custom view to a layout.

Custom Components

- You can define custom components changing the progressDrawable ...

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:andr
    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">

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyle"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:indeterminateOnly="false"
        android:progressDrawable="@drawable/custom_progress"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.5" />

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

Custom Components

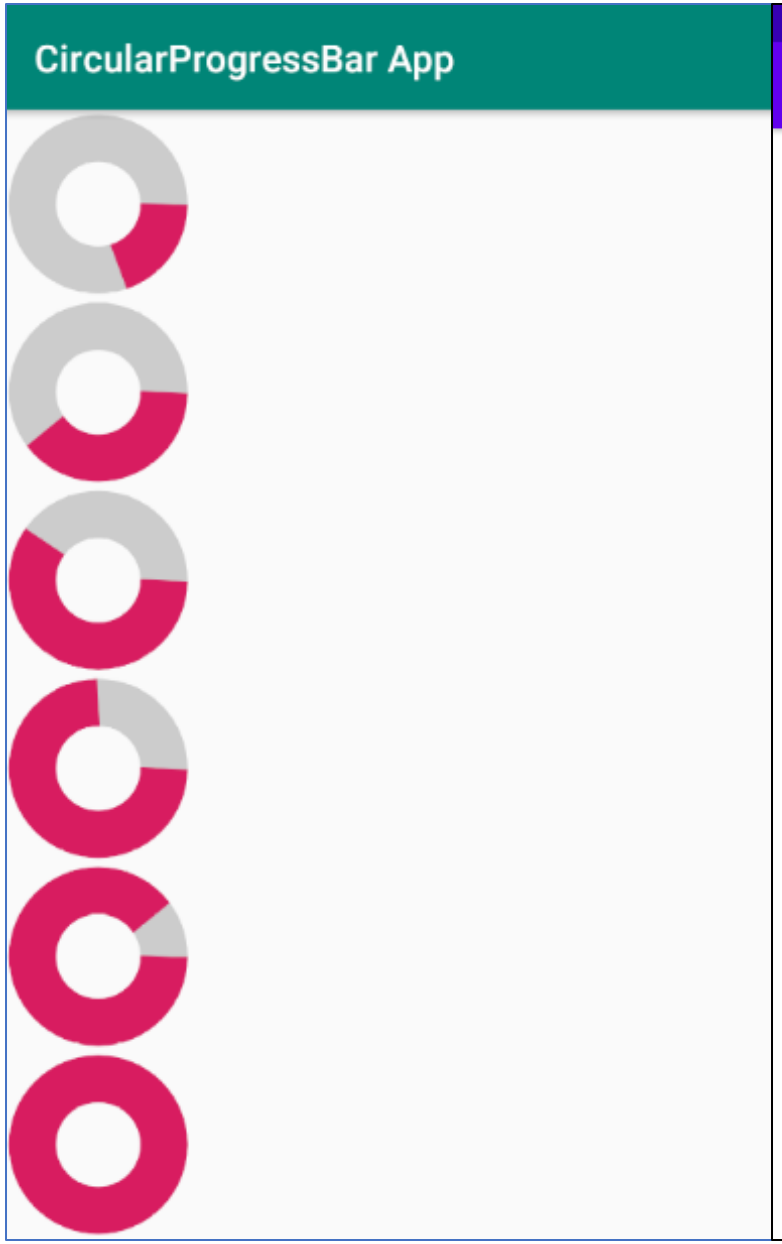
- **progressDrawable**:
  progress drawable is an attribute used in Android to set the custom drawable for the progress mode.

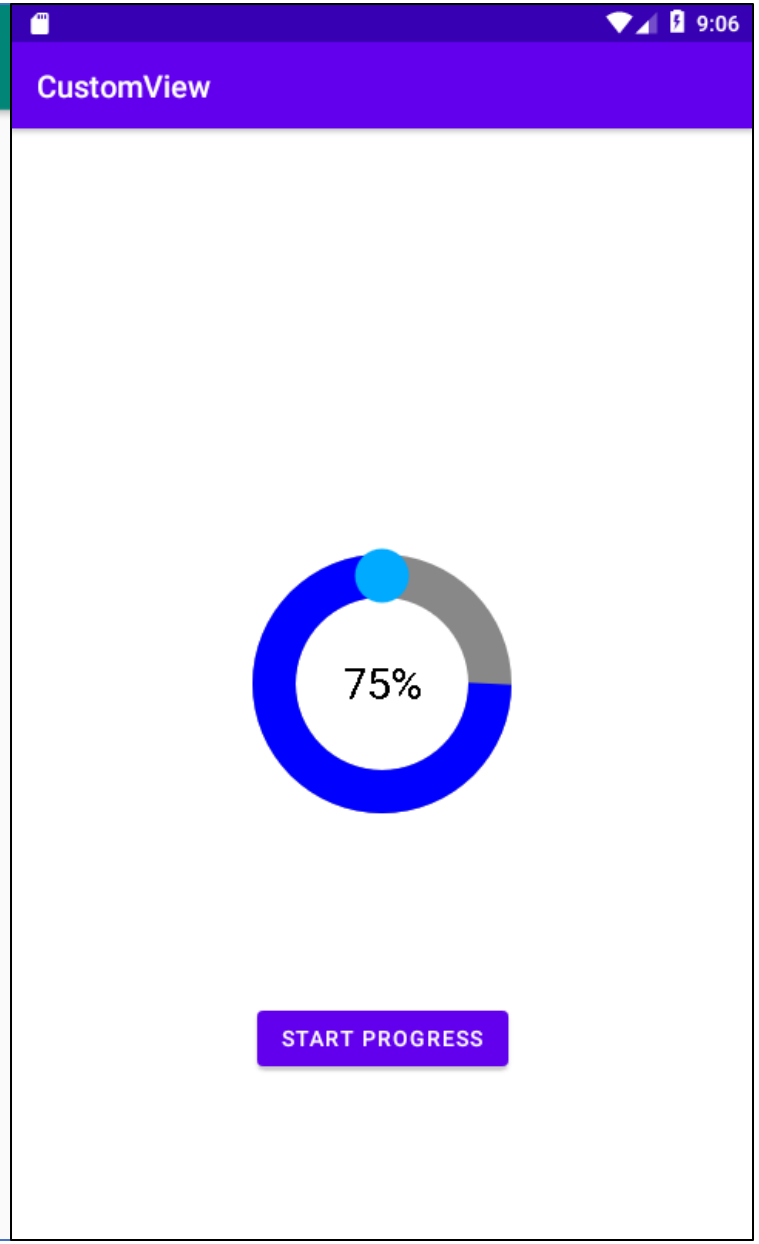
```xml
<gradient
    android:endColor="#6bf"
    android:startColor="#00f"
    android:useLevel="true" />
```

ty_main.xml | custom_progress.xml | MainActivity.k

```xml
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.andro
    <item android:id="@android:id/background">
        <shape android:shape="ring"
            android:thickness="20dp"
            android:useLevel="false">
            <solid android:color="#29AAAA"/>
        </shape>
    </item>
    <item android:id="@android:id/progress">
        <shape android:shape="ring"
            android:thickness="20dp"
            android:useLevel="true">
            <solid android:color="#3234FF"/>
        </shape>
    </item>
</layer-list>
```

Example: Custom Circular Progress Bar in Kotlin





Subclass a View

- All of the view classes defined in the Android framework extend View
- To allow Android Studio to interact with your view,
  - at a minimum you must provide a constructor that takes a **Context** and an **AttributeSet** object as parameters.
  - This constructor allows the layout editor to create and edit an instance of your view.

```
class CircularProgressBar(context: Context, attrs: AttributeSet) : View(context, attrs)
{

}
```

Android View Class Constructors

- View has four constructors and you will need to override one of them at least to start your customization.
- constructor(context: Context)
  To create a new View instance from Kotlin code, it needs the Activity context.
- constructor(context: Context, attrs: AttributeSet)
  To create a new View instance from XML.
- constructor(context: Context, attrs: AttributeSet, defStyleAttr: Int)
  To create a new view instance from XML with a style from theme attribute.
- constructor(context: Context, attrs: AttributeSet, defStyleAttr: Int, defStyleRes: Int)
  To create a new view instance from XML with a style from theme attribute and/or style resource.

Bitmap, Canvas and Paint

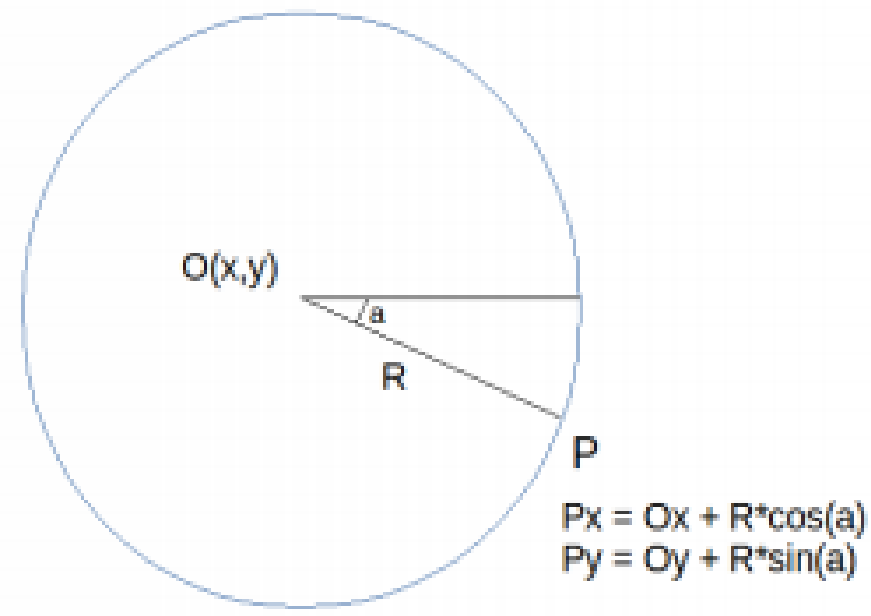
- To draw something, you need 4 basic components: you paint on a **Bitmap** surface; the **Canvas** class provides the drawing methods (primitives) to draw on a bitmap and the **Paint** class specifies how you draw on the bitmap.
  - A Bitmap to hold the pixels.
  - A Canvas to host the draw calls (writing into the bitmap) the Canvas object contains the **bitmap** on which you draw.
  - A **drawing primitive** allows you to draw something on the bitmap (e.g. Rect, Path, text, Bitmap),
  - A Paint allows you to specify the style of your drawing (colors, font, effects,…).

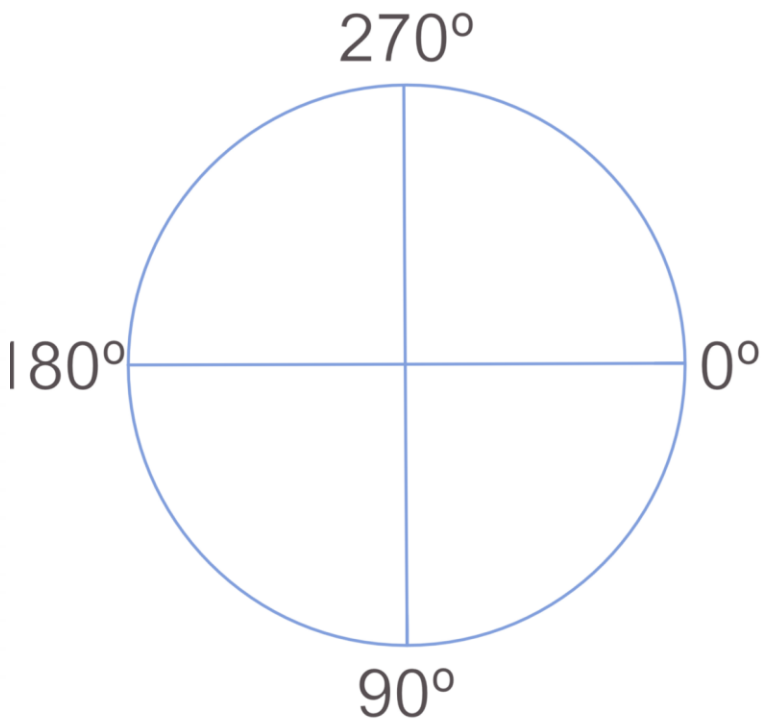
What to draw, is handled by Canvas

How to draw, is handled by Paint

Math to Draw on a circle

```
// degrees to radiants
val angle = percentage*360f * Math.PI / 180
val deltaX = radius * cos(angle).toFloat()
val deltaY = radius * sin(angle).toFloat()
canvas?.drawCircle( cx: center.x+deltaX,  cy: center.y+deltaY,  radius: radius*0.25f, paintFg2)
```





Create Drawing Objects

```
class CircleProgressBar(context: Context, attrs: AttributeSet) : View(context, attrs)
{
```

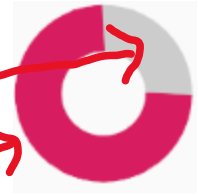
Canvas drawing on the Bitmap

```
    override fun onDraw(canvas: Canvas) {
        // let us draw by calling the method of the super class
        super.onDraw(canvas);
```

Paint

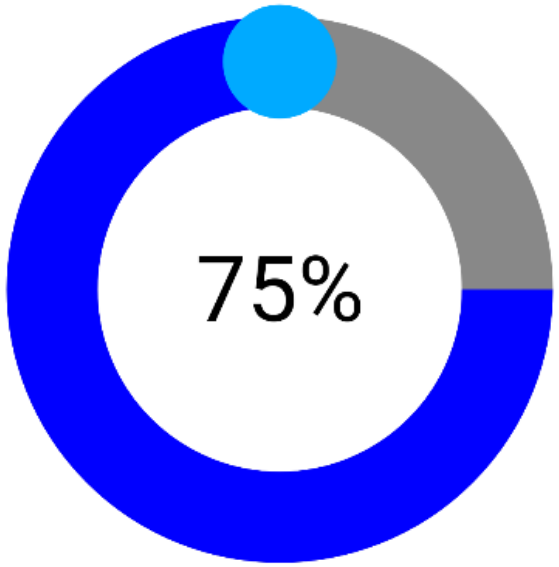
```
        paint.style = Paint.Style.STROKE
        paint.isAntiAlias = true

        val max_size = min(width, height)
        paint.strokeWidth = max_size*0.25f
        val pad = paint.strokeWidth *0.6f
        mDrawRect.set(0f+pad, 0f+pad, width.toFloat()-pad, height.toFloat()-pad)

        var startAngle = 0f;
        var drawTo = startAngle + (percentage * 360);

        paint.color = Color.LTGRAY
        canvas.drawArc(mDrawRect, 0f, 360f, false, paint)
        paint.color = ContextCompat.getColor(context, R.color.colorAccent)
        canvas.drawArc(mDrawRect, startAngle, drawTo, false, paint)
    }
}
```

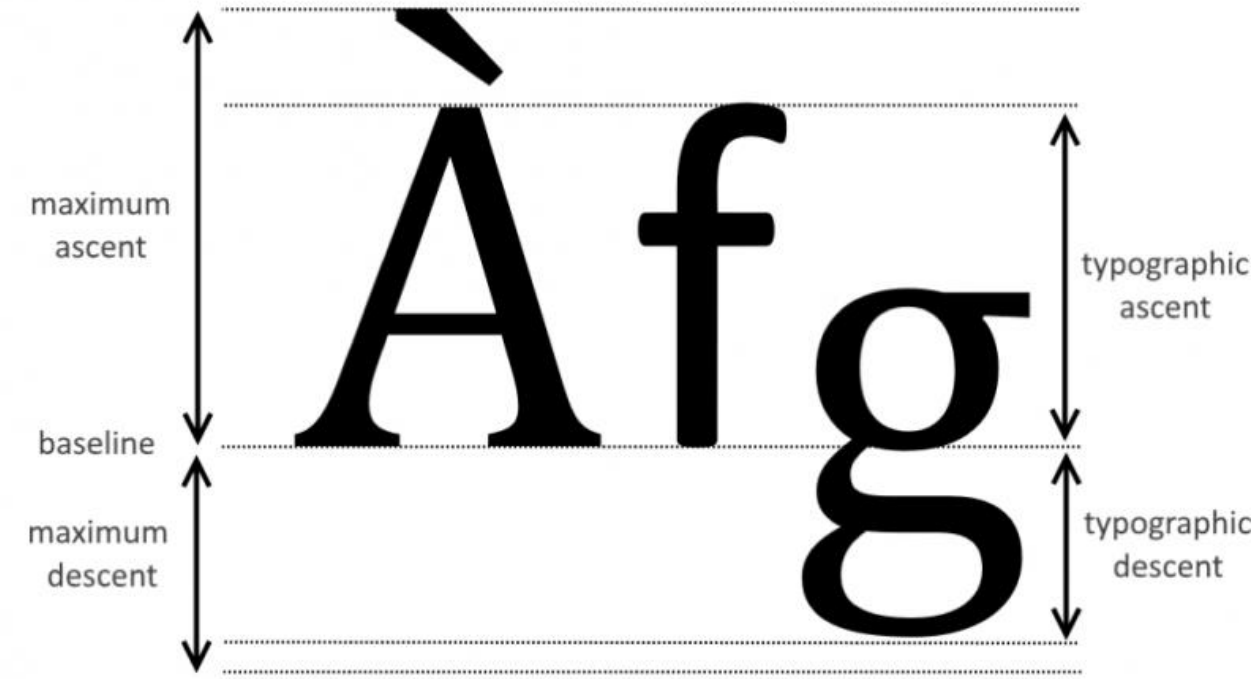
```
private val mDrawRect: RectF =
RectF(0f, 0f, 0f, 0f)
private val paint: Paint = Paint()
var percentage : Float = 0.75f
    get() = field
    set(value) {
        field = value
        invalidate()
        requestLayout()
    }
```



Center text

```
paintText.textAlign = Paint.Align.CENTER
val delta = (paintText.descent() + paintText.ascent()) / 2
canvas?.drawText("Afg", center.x, center.y-delta, paintText)
```





Create Drawing Objects

```
override fun onDraw(canvas: Canvas?) {
    super.onDraw(canvas)
    val center = PointF( x: width.toFloat()/2f,  y: height.toFloat()/2f)
    val minSize = 0.7f * min(width.toFloat(), height.toFloat())

    initPaints(minSize)

    val radius = minSize/2f
    val left = center.x-radius
    val right = left + 2*radius
    val top = center.y-radius
    val bottom = top + 2*radius

    rect.set(left, top, right, bottom)

    canvas?.drawArc(rect,  startAngle: 0f,  sweepAngle: 360f,  useCenter: false, paintBg)
    canvas?.drawArc(rect,  startAngle: 0f,  sweepAngle: percentage*360f,  useCenter: false, paintFg)

    // degrees to radiants
    val angle = percentage*360f * Math.PI / 180
    val deltaX = radius * cos(angle).toFloat()
    val deltaY = radius * sin(angle).toFloat()
    canvas?.drawCircle( cx: center.x+deltaX,  cy: center.y+deltaY,  radius: radius*0.25f, paintFg2)

    val delta = (paintText.descent() + paintText.ascent()) / 2
    canvas?.drawText( text: "${(percentage*100).toInt()}%", center.x,  y: center.y-delta, paintText)
}
```

```
private fun initPaints(minSize:Float){
    paintBg.strokeWidth = minSize*0.2F
    paintFg.strokeWidth = minSize*0.2F
    paintFg2.strokeWidth = minSize*0.2F
    paintText.textSize = minSize*0.2F
    paintText.textAlign = Paint.Align.CENTER
}
```

```
init {
    paintBg.style = Paint.Style.STROKE
    paintBg.isAntiAlias = true
    paintBg.color = Color.GRAY

    paintFg.style = Paint.Style.STROKE
    paintFg.isAntiAlias = true
    paintFg.color = Color.BLUE
    // paintFg.strokeCap = Paint.Cap.ROUND

    paintFg2.style = Paint.Style.FILL
    paintFg2.isAntiAlias = true
    paintFg2.color = Color.parseColor( colorString: "#00AAFF")

    paintText.color = Color.BLACK
}
```

```
class CircleProgressBar(context: Context, attrs: AttributeSet): View(context,  attrs) {
    val rect = RectF( left: 0f,  top: 0f,  right: 0f,  bottom: 0f)
    val paintBg = Paint()
    val paintFg = Paint()
    val paintFg2 = Paint()
    val paintText = Paint()
    var percentage = .75f
        set(value) {
            field = min(value,  b: 1f)
            invalidate()
            requestLayout()
        }
```

Add custom view to the layout

- Now you can add your custom view in your layout by adding the following lines to the XML layout

```
 xml version="1.0" encoding="utf-8"
LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.cotools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 tools:context=".MainActivity"

 <!--Full path for the custom view -->
   it.uninsubria.pdm.circularprogressbarapp.CircleProgressBar
    android:id="@+id/circularProgressbar"
    android:layout_width="100dp"
    android:layout_height="100dp"

 LinearLayout
```

Setup the progress bar

- Change the percentage and draw Custom View

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        circularProgressbar.percentage = 0.75f
    }
}
```

Animate the progress Bar

```
class CustomProgressActivity : AppCompatActivity() {
    lateinit var countDown: CountDownTimer
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_custom_progress)

        button2.setOnClickListener{  it: View!
            progressCircular.percentage = 0f
            countDown.start()
        }

        countDown = object: CountDownTimer( millisInFuture: 11*1000,  countDownInterval: 1000) {
            override fun onTick(ms: Long) {
                progressCircular.percentage += 0.1f
            }

            override fun onFinish() {
            }

        }
    }
}
```

Add an Event Listener

```
setOnClickListener {  it: View!
    percentage = 0f
}
setOnLongClickListener{  it: View!
    object:CountDownTimer( millisInFuture: 11*1000,  countDownInterval: 1000) {
        override fun onTick(ms: Long) {
            percentage += 0.1f
        }
        override fun onFinish() {
        }
    }.start()
    true  ^setOnLongClickListener
}
```

Define Custom Attributes

- Well-written custom views can also be added and styled via XML.
- To enable this behavior in your custom view, you must:
  - Define custom attributes for your view in a <declare-styleable> resource element
  - Specify values for the attributes in your XML layout
  - Retrieve attribute values at runtime
  - Apply the retrieved attribute values to your view

Define Custom Attributes

- In `res/values/attrs.xml`

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="CircleProgressBar">
        <attr name="bgColor" format="reference|color" />
        <attr name="fgColor" format="reference|color" />
        <attr name="endColor" format="reference|color" />
    </declare-styleable>
</resources>
```

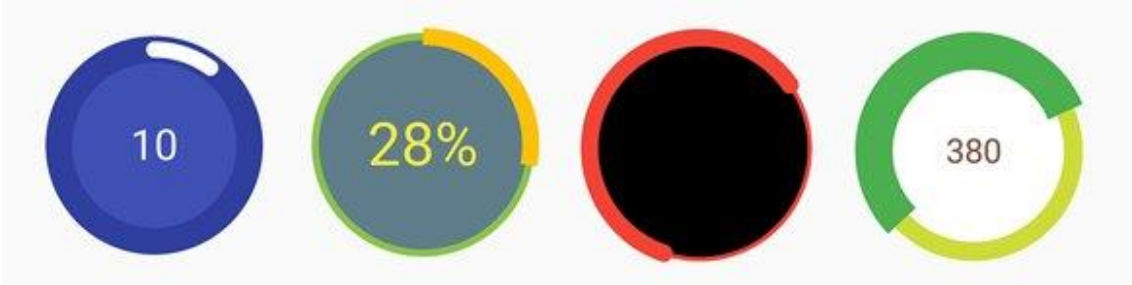
```xml
<it.insubria.customprogressbardrawable.CircleProgressBar
    android:id="@+id/progressCircular"
    android:layout_width="200dp"
    android:layout_height="200dp"
    app:bgColor="@color/design_default_color_secondary"
    app:fgColor="@color/purple_500"
    app:endColor="@color/black"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

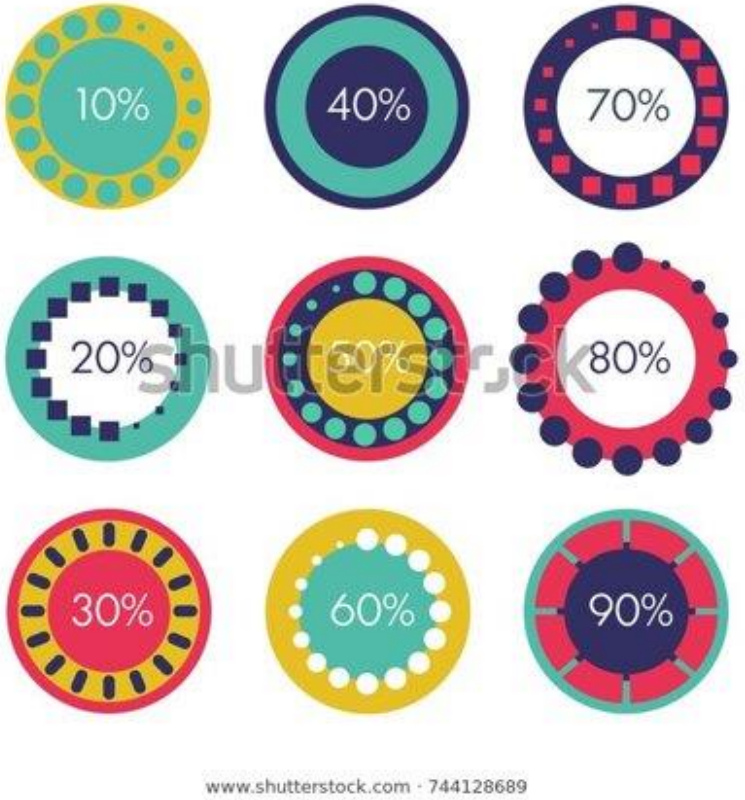
Retrieve attribute values at runtime

- In `init{}` …

```
// Get the custom attributes
val typedArray = context.obtainStyledAttributes(
    attrs, R.styleable.CircleProgressBar,  defStyleAttr: 0,  defStyleRes: 0
)
// Set the colors from the attribute values.
paintBg.color = typedArray.getColor(R.styleable.CircleProgressBar_bgColor, Color.GRAY)
paintFg.color = typedArray.getColor(R.styleable.CircleProgressBar_fgColor, Color.BLUE)
paintFg2.color = typedArray.getColor(R.styleable.CircleProgressBar_endColor,
    Color.parseColor( colorString: "#00AAFF"))
typedArray.recycle()
```

Other Circular Progress Bar





The CustomFanController app

developer.android.com