# B.1- Nozioni di base di layout

Android utilizza dei widget chiamati VIEWs per visualizzare l’interfaccia. Tutti i widget sono sottoclassi di una classe antenato comune, chiamata appunto View.

Tutte le view devono avere width e heigth definite affinchè siano visibili, e possiedono un campo id che permette di identificarle nel’app.

Una importante sottoclasse di View è ViewGroup, che è l’antenato di tutte le classi Layout: classi invisibili il cui ruolo è contenere e ordinare delle View.

Nello sviluppo di un’app è possibile creare layouts e views in 3 maniere:

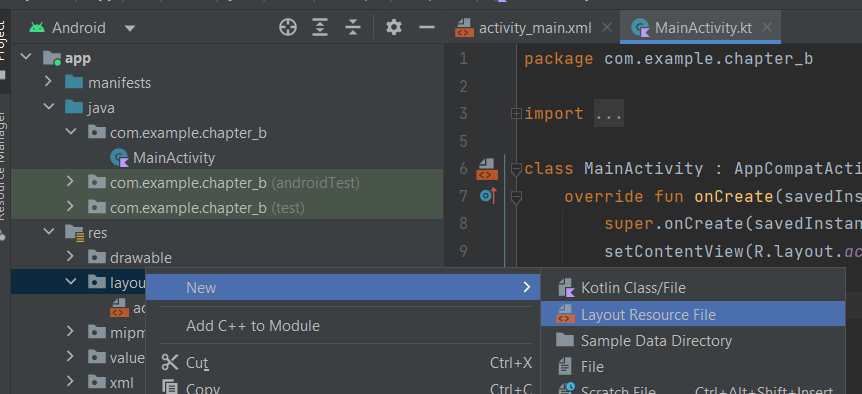
* Programmaticamente, utilizzando i loro costruttori
* Con dei file risorse XML
* Con una libreria dichiarativa chiamata Jetpack Compose

In questa guida utilizzeremo il **formato** **XML** per definire i layout, in quanto attualmente il più utilizzato.

## Creare un file di layout

I file di layout sono contenuti all’interno della directory *res/layout*. Usando il menu contestuale (tasto destro) è possibile crearne di nuovi. Il popup risultante ci chiederà il nome e la classe *layout* di base, che di default è *ConstraintLayout*

Il nome del file layout deve essere scritto in **snake case** e segue la convenzione tipo\_nome

 Graphical user interface, application

Description automatically generated

Nella parte *qualifiers* è possibile inserire dei parametri che restringeranno l’utilizzo di quel layout per specifici dispositivi.

Esempio: solo dispositivi in modalità Night e che hanno il testo impostato a RightToLeft

Graphical user interface, application

Description automatically generated

## Constraint Layout

Il Constraint Layout è un ViewGroup che lavora sul concetto di ordinare le view che contiene assegnando loro dei valori *constraint* per allineare ogni elemento rispetto a un altro. Le view contenute saranno tutte sullo stesso livello di parentela (*flat layout*), riducendo notevolmente la complessità rispetto a ordinamenti innestati.

Ad esempio, una pagina può contenere un titolo allineato al bordo in alto a sinistra del layout che lo contiene. Nella stessa pagina, un paragrafo può essere allineato al bordo di sinistra del layout, e verticalmente al di sotto del titolo.

## DP, margin e padding

**DP** è l’unità di misura delle dimensioni delle View. Significa Density Indipendent Pixels, perché si ridimensiona automaticamente in base alla densità dello schermo su cui viene visualizzata la view, permettendo di poter costruire solo 1 layout per diverse grandezze di device.

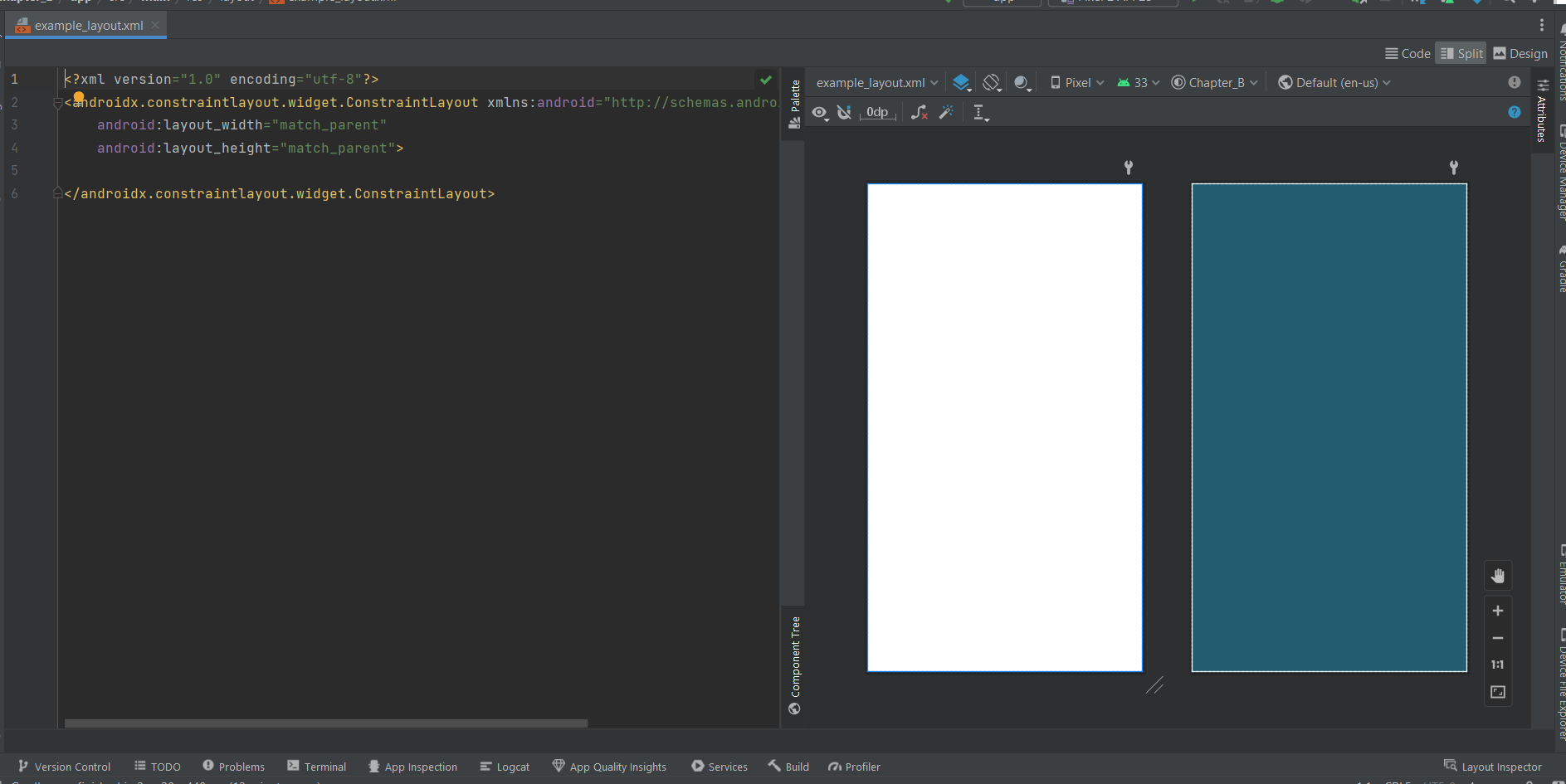
**Padding** è la distanza che la view interpone tra i suoi bordi e il suo contenuto.

**Margin** è la distanza che la view interpone tra i suoi bordi e quelli delle view vicine. Interagisce solo con le view che hanno il suo stesso grado di parentela o maggiore. Nel constraint layout, questa distanza è applicata ai constraint che identificano dove è posizionata la view.

Ad esempio, utilizzando il titolo di prima, un Margin di 24dp creerà una distanza tra il titolo ed il bordo alto-sinistra a cui è allineato.

## Editor Grafico e Testuale

Doc: https://developer.android.com/studio/write/layout-editor

Android Studio ci fornisce un editor di Layout sia grafico che testuale.  


Utilizzando i pulsanti *Code, Split, Design* è possibile cambiare impostazione dell’editor lavorando sempre sullo stesso file.

*Design* è utilizzato più spesso per dare un’impostazione generale, mentre la parte *Text* permette il fine-tuning e il lavoro su layout complessi.

## Aggiungere elementi: Design

Selezionare l’impostazione *Design.*

Trascinare un oggetto TextView dalla tavolozza in alto a sinistra dell'area di progettazione nell'editor di layout e rilasciarlo.

TextView è un widget il cui scopo è mostrare testo.

Graphical user interface, application

Description automatically generated

Una TextView sarà aggiunta, notare un punto esclamativo rosso nel Component tree.

Posizionando il puntatore sul punto esclamativo si visualizzerà un messaggio di avviso dicendo che la view non ha dei *constraint*, quindi il layout non sa dove ordinarla.

Per creare dei constraint, nel pannello a destra Layout > Constraint Widget

A picture containing scatter chart

Description automatically generated

Cliccare sul + in alto, e selezionare 24 nel menu a tendina che comparirà. Ripetere per il + sinistro

Questo creerà due constraint che allineeranno il testo in alto a sinistra, con un *margin* di 24dp dall’elemento a cui ci si sta allineando.

Timeline

Description automatically generated

Questi constraint verranno creati allineandosi al padre della view, il ConstraintLayout

Text

Description automatically generated

Ogni view ha 4 lati: Start, End, Top, Bottom. Ognuno di questi lati puo’ essere allineato al lato di un’altra view.

Selezionando *Code* nell’editor potremo vedere come questi dati vengono convertiti a codice, creando margini e constraint.

Text

Description automatically generated

E’ anche possibile creare constraint selezionando una view e trascinando i punti vuoti verso il bordo di altre view.

## Aggiungere elementi: Code

Utilizzando l’editor *Split*, andiamo ad aggiungere un altro campo di testo.

Sotto la textView che già abbiamo, aggiungere delle righe e digitare **<TV**   
Si proporrano delle scelte di code completion:

Graphical user interface, text, application

Description automatically generated

Cliccando il suggerimento il nostro widget verrà parzialmente precompilato, con i campi width e height.

Scriviamo una TextView con le caratteristiche uguali a quella precedente, ma con id e testo diverso:  
Text

Description automatically generated

La nuova TV, textView2, è solo parzialmente visibile (il numero 2 grigio nella preview): è posizionata esattamente come quella creata in precedenza. Ha infatti gli stessi constraint e gli stessi margin.   
Aggiungiamo un background verde e modifichiamo il testo in Avocado:  


Ora vediamo la seconda textView ma non la prima. Questo perché quando più elementi si sovrappongono, *vengono* *ordinati in base a come sono dichiarati nel file XML*: textView è scritta prima, quindi andrà sotto textView2.

Ora spostiamo textView2 sulla destra di textView in modo da vederle entrambe.   
Sostituiamo il constraint *layout\_constraintStart\_toStartOf* con un constraint che allinea lo *start* di textView2 all’*end* di textView. Il constraint da utilizzare sarà *layout\_constraintStart\_toEndOf*.   
Puoi utilizzare le lettere iniziali della definizione *Constraint Start To End* e premere invio per scriverlo rapidamente.

A screenshot of a computer

Description automatically generated

Infine, inseriamo l’id *textView* per specificare all’end di quale view allinearsi.

Avremo cosi’ allineato textView2 sulla sinistra di textView  
Graphical user interface, text

Description automatically generated

## Best Practice: no hardcoding

L’IDE ci sta sgridando con un warning, dicendo che le nostre stringhe sono hardcodate:

Graphical user interface, text, website

Description automatically generated

È buona norma non hardcodare risorse in generale, ma nel caso specifico delle stringhe è necessario: avere un file con tutte le stringhe serve per poter tradurre la nostra app in altre lingue.

Selezioniamo la linea di testo con il warning e *attiviamo l’assistente (alt-invio) > extract string resource*

A screenshot of a computer

Description automatically generated

Diamo un nome alla nostra nuova risorsa e salviamo  
Graphical user interface, application

Description automatically generated  


La stringa sarà salvata e il nostro layout automaticamente aggiornato con il riferimento.   
Controlliamo che tutto sia a posto andando a leggere il file di stringhe: nel codice, ctrl-click (o middle mouse btn) sul nome della stringa.

Questo ci porterà a *res/values/strings.xml* dove vengono salvate tutte le stringhe:

Graphical user interface, text

Description automatically generated

E’ buona norma non hardcodare valori che possano essere contenuti in file di risorse – come stringhe, dimensioni e colori.

## Larghezza e Altezza widget

*layout\_width e layout\_height* possono essere definiti in più maniere:

* *Valore fisso espresso in dp*: ad esempio 100dp
* *wrap\_content:* la view si ridimensionerà al minimo indispensabile per visualizzare il contenuto
* *match\_parent:* la dimensione sarà pari alla stessa dimensione del layout contenitore
* *0dp:* questa opzione, disponibile solo per view all’interno di un Constraint Layout, sostituisce *match\_parent.* La view si ridimensionerà per raggiungere la massima ampiezza consentita dai suoi constraint, che devono entrambi essere settati per quella direzione – verticale o orizzontale.

## Immagini

Tramite l’editor *Design*, inseriamo una ImageView:

*A screenshot of a computer

Description automatically generated with medium confidence*

Alla richiesta di quale immagine visualizzare, selezioniamo la prima dalla lista *sample data*  
Graphical user interface

Description automatically generated

Definiamo i constraint dell’immagine legando start, end e bottom al layout. Infine trasciniamo top al bottom di textView

Diagram

Description automatically generated

L’immagine ora sarà centrata nello spazio che intercorre tra il layout e il bottom di textView.  
Diagram

Description automatically generated with low confidence

La nostra immagine funziona, ma in realtà avendo selezionato un’immagine dalla galleria “sample data” l’editor ha inserito un’immagine con una modalità che la rende visibile solo in fase di progettazione. Se lanciamo l’app con la freccia verde, l’immagine non comparirà nel nostro device.

Graphical user interface, application

Description automatically generated

*La chiave inglese mostra i campi le cui modifiche sono visibili solo in progettazione.*

Aggiungiamo un nuovo obbiettivo:  
questa immagine deve essere un banner che mostra un paesaggio, e deve essere visibile sotto il testo dell’esempio precedente.

Passiamo alla modalità *Split.*

Aggiungiamo l’immagine da visualizzare: importiamo un file .jpg nel progetto.   
Apri Resource Manager > pulsante + > Import Drawables. Seleziona il file dal tuo pc e accetta le opzioni di default nei menu successivi.

Graphical user interface, text

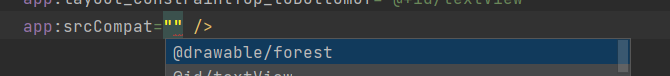
Description automatically generated

*Chiudi e riapri resource manager per aggiornarlo (bug del’IDE)*. Ora la nostra immagine sarà visibile nelle risorse. È stata inserita nella dir **res/drawable** dove vengono mantenute la maggior parte delle immagini dell’app.

E’ anche possibile inserire un file da windows con copia-incolla.   
ctrl-c sul file in windows. Su android studio, selezionare la directory di destinazione dall’albero Project, poi ctrl-v.

Ora andiamo a inserire l’immagine effettiva nell’ImageView

Sostituiamo la riga tools:srcCompat con app:srcCompat, nel valore selezionando il nome del file che abbiamo importato.

Il file immagine viene riconosciuto come *drawable* nelle risorse del progetto quando presente nella directory *res*/*drawable*. Non tutti i tipi di immagini sono supportati.  


Ora verrà visualizzata la nostra immagine, ma c’è un problema. Essendo imageView impostata con height e width *wrap\_content*, si andrà a ridimensionare in base alle dimensioni dell’immagine che gli abbiamo fornito. In questo caso, l’immagine è più grande della risoluzione dello schermo, e imageView diventa troppo grande:

Graphical user interface, application

Description automatically generated

Sebbene di base imageView mostri l’immagine scalandola per le dimensioni dello schermo, la dimensione del widget è pari alle dimensioni dell’immagine originale.  
Modifichiamo quindi la sua altezza in 100dp e larghezza in 0dp: questo metterà un limite a quanto può espandersi.

ImageView cerca sempre di mantenere le corrette proporzioni dell’immagine

il risultato sarà poco elegante:

Graphical user interface, text, application

Description automatically generated

Per ottenere un banner, aggiungiamo il parametro scaleType="centerCrop" . Inoltre rimuoviamo constraint\_bottom cosi’ da rimuovere il centraggio e legarla solo a textView verso top. Infine, aggiungiamo marginTop 24dp.  
Text

Description automatically generated

Abbiamo raggiunto il nostro obbiettivo, ma come ultimo controllo decidiamo di dare un’occhiata a cosa succede quando l’utente ruota il device in posizione landscape:

Graphical user interface, application

Description automatically generated

Sebbene diverso, il nostro banner è ugualmente apprezzabile.  
Graphical user interface

Description automatically generated

## Best Practice: resource naming

Aggiungiamo un obbiettivo:  
Il titolo deve rappresentare lo scontro mortale tra due frutti – ed esempio: Mela vs Banana

Modifichiamo l’id della prima textView per riflettere la sua nuova funzionalità. L’id è un campo che è utilizzato come riferimento da altre view – nel nostro esempio, nei constraint di textView2 e imageView. Potremmo modificare tutto a mano, ma Android Studio ci viene in soccorso con il *refactor > rename*

Text

Description automatically generated

Selezionando l’opzione – ma è più pratico *shift-f6* – è possibile rinominare il campo e aggiornare tutti i punti del codice che ne fanno riferimento.

Graphical user interface, text, application, chat or text message

Description automatically generated

L’ID è una risorsa che identifica la view nell’app, e deve essere specifico e parlante.  
Una convenzione è: cosa contiene \_ una identificazione tra elementi simili \_ tipo di widget.  
Android non richiede ID univoci.  
  
Così anche solo leggendo l’id possiamo avere un’idea di che widget sia.

Rinominiamo anche textView2 in fighter\_two\_tv, e imageView in fight\_bg\_iv.

Facciamo delle modifiche ai nostri titoli:

* Rinomina anche textView2 in fighter\_two\_tv, e imageView in fight\_bg\_iv.
* fighter\_one\_tv: cambia il testo in Apple e il bg in Holo red light  
  Text

  Description automatically generated
* Aggiungi tra fighter\_one e fighter\_two una nuova TV *fight\_versus\_tv* , che conterrà il testo “VS”.   
  Imposta il constraint orizzontale all’end di fighter\_one\_tv  
  Imposta il constraint verticale *baseline* al *baseline* di fighter\_one\_tv.



layout\_constraintBaseline\_toBaselineOf allineerà verticalmente la base del testo di due widget.

* Modifica *fighter\_two\_tv* per legare il suo start all’end di *fight\_versus\_tv*  
  Text

  Description automatically generated

Risultato:  
Graphical user interface, application, website

Description automatically generated

## Chains

Finora i campi di testo sono allineati sulla sinistra dello schermo: se volessimo centrare *entrambi* i campi, mantenendo comunque l’ordine di uno prima dell’altro, incomberemmo in alcuni limiti: uno o entrambi, sarebbero sempre decentrati.   
Le *Chains* risolvono questo problema!

Una Chain è una serie di widget legata strettamente uno all’altra, con un ordinamento in comune.

Decidiamo di centrare i tre campi di testo nello spazio orizzontale: costruiamo una Chain.

Nella parte grafica dell’editor *Split,* trascina e seleziona i tre campi di testo. Poi tasto destro > chains > create horizontal

Graphical user interface, application

Description automatically generated

Dando un’occhiata al codice, questo comando modifica *tutti i widget della catena* impostando *entrambi* *i constraint* start e end al widget precedente e successivo.

Text

Description automatically generated

Il risultato è promettente, ma troppo separato nello spazio:

A picture containing graphical user interface

Description automatically generated

Modificando lo stile della chain possiamo scegliere tra alcune opzioni per come vengono organizzati gli spazi. E’ possibile utilizzare sempre lo stesso menu contestuale per farlo, ma noi andremo nell’editor di testo.

Il widget responsabile del setup di una chain è il primo della chain nell’xml.

Nel nostro caso, fighter\_one\_tv – è il primo, i cui constraint start sono attaccati a *parent*. Andiamo ad aggiungere un chainStyle:

app:layout\_constraintHorizontal\_chainStyle="packed"

Graphical user interface, application, website

Description automatically generated

Le chains possono essere ordinate in : packed, spread, spread\_inside

Ruotando il layout in modalità *portrait*, vedremo che le view si riorganizzano in base allo spazio disponibile.

Graphical user interface

Description automatically generated

Per ora tutto è un po’ spostato a sinistra: questo perché Apple e Avocado sono parole con larghezza diversa, e le nostre TV sono impostate con larghezza wrap\_content.

Facciamo queste modifiche a *fighter\_one\_tv* e *fighter\_two\_tv*:

* Rimuovi marginStart
* Aggiungi:  
  android:minWidth="150dp"  
  android:textAlignment="center"

E questa modifica a *fight\_versus\_tv*, per separarla dalle altre 2:

* Aggiungi  
  android:layout\_marginEnd="24dp"

Ora i widget sono centrati   
Graphical user interface, application

Description automatically generated

## Esercizio: Postcard o Meme

Sviluppa un’app con le seguenti caratteristiche:

* Una singola pagina, con un’immagine di sfondo a schermo intero
  + Crea un nuovo progetto con una *Empty Activity* e scrivi l’interfaccia nel file layout *activity\_main.xml*.
* Deve contenere almeno due campi di testo, il cui contenuto deve essere facilmente leggibile sullo sfondo
  + TextView ha un parametro *textColor.* Android fornisce alcuni colori di sistema con riferimento *@color/…* e *@android:color/…*
  + Uno dei due campi deve essere scritto in *Italic*
* Deve contenere almeno una chain
* No hardcoding: né di stringhe, né di dimensioni (dp)
* ID parlanti