**Apunte Módulo 1 - Unidad 3**

**1 - Widgets.**

Kivy usa la palabra widget para describir cualquier elemento de interfaz de usuario, como puede ser la etiqueta Label introducida en el ejercicio anterior, los campos de entradas de texto, o elementos más complejos utilizados para representar películas.

Podemos pensar en un widget como una especie de caja que tiene comportamientos y puede contener otros elementos. La clase de Widget es la más básica de este tipo, y si bien se encuentra vacía todos los otros widgets extienden de esta clase.

La caja contenedora principal es denominada en la literatura como widget “root” o raíz. Debe existir un único widget raíz en nuestro código Kv. En el ejemplo de la sección anterior podemos ver que el widget Label el elementos que se encuentra más a la izquierda en nuestro código y que el contenido del mismo al igual que en python se encuentra tabulado para indicar que se trata de un bloque de contenido dentro del mismo.

Un ejemplo un poco más interesante lo puede constituir el ejercicio “segundaapp” en donde utilizamos un layout predefinido para crear una distribución de tres textos en pantalla, aquí el widget raíz es

|  |  |
| --- | --- |
| **segundaapp/main.kv** |  |
| BoxLayout:      Label:          text: "Pera"      Label:          text: "Manzana"      Label:          text: "Naranja" | A screenshot of a computer  Description automatically generated |

**Tarea:** Ingrese a la página de kivy, encuentre la clase a partir de la cual se pueden crear botones, y cree una aplicación similar a la del bloque temático 3 pero que contenga tres botones en lugar de instancias de la clase Label.

**Nuestros propios Widgets.**

Entrando un poco más en detalles, es posible crear un widget propio para nuestra aplicación en lugar de utilizar las clases propias de Kivy, veamos un ejemplo:

|  |  |
| --- | --- |
|  | **terceraapp/main.py** |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | from kivy.app import App  from kivy.lang import Builder  from kivy.uix.widget import Widget    class MiWidget(Widget): pass    class MiApp(App):      def build(self):          return MiWidget()    if \_\_name\_\_ == '\_\_main\_\_':      MiApp().run() |

En el código anterior lo que estamos haciendo es crear un Widget propio llamado “MiWidget” el cual hereda de la clase Widget de Kivy, y luego retornamos una instancia de dicha clase dentro de nuestra app, en este caso “MiApp”. Luego el código mi.kv se ve modificado ya que ahora indicaremos el contenido de la vista asociada a la clase “MiWidget” de la siguiente manera:

|  |  |
| --- | --- |
|  | **terceraapp/mi.kv** |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11** | <MiWidget>:      Label:          text: 'Pera'          pos: 0, 0        Label:      text: 'Manzana!'      font\_size: 28      pos: 100,100      color: .6,1,.2,1      size: 100, 50 |

Cómo podemos ver al ejecutar el ejemplo anterior dentro de nuestra caja, los elementos se ubican considerando un plano 2D en donde el origen de coordenadas se encuentra en el margen inferior izquierdo. En el código, la clase que da origen a la vista se pone entre los signos de menor y mayor sin aplicar indentación <>, y a continuación se agregan las instancias de los diferentes widgets de Kivy junto con la declaración de los distintos atributos.

A screenshot of a computer

Description automatically generated with medium confidence

Cuando trabajamos directamente con la clase widgets las coordenadas las podemos dar en coordenadas absolutas, en px. Sin embargo también podemos utilizar coordenadas relativas, realicemos una modificación al código anterior para observar este punto.

|  |  |
| --- | --- |
|  | **cuartaapp/mi.kv** |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | <MiWidget>:      Label:          text: 'Pera'          pos: root.x, root.top - self.height        Label:          text: 'Manzana!'          pos: root.right - self.width, root.top - self.height          size: 100, 50          color: .6,1,.2,1          font\_size: 28 |

Aquí como es de esperar self está haciendo referencia al widget utilizado, por lo que en código anterior siempre estamos restando la altura del “widget Label”, mientras que root está haciendo referencia a la clase widget que se encuentra definida en la parte más alta de la jerarquía.

A screen shot of a computer

Description automatically generated with medium confidence

De esta forma podemos realizar una aplicación responsive.

En resumen:

|  |  |
| --- | --- |
| **pos** | **Descripción** |
| **root.top** | Distancia superior con relación al elemento raíz |
| **root.right** | Distancia desde la derecha con relación al elemento raíz |
| **root.left** | Distancia desde la izquierda con relación al elemento raíz |
| **root.bottom** | Distancia desde abajo con relación al elemento raíz |
| **self.height** | Altura del widget |
| **self.width** | Ancho del widget |

**Tarea:** Cree una aplicación que se vea como la siguiente imagen y que se adapte cuando variamos el tamaño de pantalla.

A picture containing text, screenshot, font, number

Description automatically generated

**Atributos compartidos.**

En el caso de que varios widgets del mismo tipo compartan los mismos  atributos, podemos crear un widget personalizado dentro de nuestro archivo kv que contenga dichos atributos, y luego llamar diferentes instancias de esta nuevo widget en el cual indiquemos aquellos atributos que son característicos del mismo. Para realizar esto utilizamos el símbolo de arroba “@” para indicar de qué tipo de widget de kivy estamos heredando.

|  |  |
| --- | --- |
|  | **quintaapp/mi.kv** |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15** | <MiLabel@Label>:      size: 180, 50      font\_size: 30    <MiWidget>:        MiLabel:          text: 'Pera'          pos: root.x, root.top - self.height          color: 1,.6,.2,1        MiLabel:          text: 'Manzana!'          pos: root.right - self.width, root.top - self.height          color: .6,1,.2,1 |

En este ejemplo podemos ver como ambas instancias de “MiLabel” comparten tanto el tamaño de letra como la fuente pero difieren en el contenido de texto, su color y ubicación.

A picture containing text, screenshot, font, number

Description automatically generated

**Sintaxis especial.**

Hay dos sintaxis especiales para definir valores para todo el contexto Kv:

Para acceder a los módulos y clases de python desde kv,

|  |  |
| --- | --- |
| **En python** | **Equivalente en kivy a:** |
| **from x.y import z as name**  **from os.path import isdir**  **import numpy as np** | #:import name x.y.z  #:import isdir os.path.isdir  #:import np numpy |

Para establecer un valor global.

|  |  |
| --- | --- |
| **En python** | **Equivalente en kivy a:** |
| **name = value** | #:set name value |

Veamos un ejemplo en el cual importaremos un módulo para poder trabajar con colores en formato hexadecimal, esta forma de indicar formatos es utilizada por diferentes plataformas como Photoshop y Gimp2, así como por el lenguaje de estilos web css. Modificaremos un poco nuestro ejercicio anterior:

|  |  |
| --- | --- |
|  | **sextaapp/mi.kv** |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | #:import C kivy.utils.get\_color\_from\_hex  <MiLabel@Label>:      size: 180, 50      font\_size: 30    <MiWidget>:        MiLabel:          text: 'Pera'          pos: root.x, root.top - self.height          color: C('#ff0000')        MiLabel:          text: 'Manzana!'          pos: root.right - self.width, root.top - self.height          color: .6,1,.2,1 |

Los números hexadecimales van del cero al nueve y de la “a” hasta la “f”, la notación de color en formato hexadecimal utiliza seis números hexadecimales precedidos por un signo de numeral, en donde los dos primero colores establecen el canal rojo, los dos siguientes el canal verde y los últimos dos el canal azul. Un valor de cero indica el canal casi apagado, y un valor de f es el valor más alto que puede tomar.

A modo de ejemplo, el siguiente valor representa un color rojo:   #ff0000

A picture containing text, screenshot, font, display

Description automatically generated

**Tarea:** Realice una búsqueda de módulos que podría ser interesante agregar desde el lenguaje kv y si se anima realice un ejercicio de ejemplo.

**Analogía de kv con python.**

Ya hemos realizado algunos ejemplos introductorios por lo que podemos analizar un poco más de cerca el lenguaje kv haciendo una analogía con python. Consideremos el siguiente código:

|  |
| --- |
| **archivo.kv** |
| MiWidgetRaiz:      BoxLayout:          Button:          Button: |

En este ejemplo definimos nuestro widget raíz, como una instancia de “MiWidgetRaiz” (no debemos utilizar caracteres en español para la asignación de nombres), el cual tiene dentro una instancia de uno de los posibles tipos de layouts (plantillas de formato predefinidas) de kivy que veremos a partir de la unidad 3. A su vez el layout “BoxLayout” tiene dentro dos instancias de la clase “Button” para definir dos botones.

Si estuviéramos trabajando en python lo podríamos haber expresado de la siguiente forma:

|  |
| --- |
| **archivo.kv** |
| root = MiWidgetRaiz()  box = BoxLayout()  box.add\_widget(Button())  box.add\_widget(Button())  root.add\_widget(box) |

Al igual que en python podemos también pasar argumentes a nuestros widgets como podremos ir viendo a lo largo de las siguientes unidades.

**Categoría de Widgets.**

Los widgets de kivy pueden agrupados en las siguientes categorías:

|  |  |  |
| --- | --- | --- |
| **Grupo** | **Widgets** | **Descripción** |
| **UX widgets** | Label, Button, CheckBox, Image, Slider, Progress Bar, Text Input, Toggle button, Switch, Video | Son Widgets de interfaz de usuario que se pueden instanciar en la interface de usuario y a partir de los cuales se pueden crear widgets más complejos. |
| **Layouts** | Anchor Layout, Box Layout, Float Layout, Grid Layout, PageLayout, Relative Layout, Scatter Layout, Stack Layout | Es un layout de diseño que permite agregar widgets de una manera predefinida. |
| **Complex UX widgets** | Bubble, Drop-Down List, FileChooser, Popup, Spinner, List View, TabbedPanel, Video player, VKeyboard, | Widgets resultantes de de combinar múltiples widgets clásicos. Se denominan como complejos porque su ensamblaje y uso no es tan genérico como el de los widgets clásicos. |
| **Behaviors widgets** | Scatter, Stencil View | Estos widgets no se renderizan, en su lugar actúan como instrucciones de interacción. |
| **Screen manager** | Screen Manager | Administra pantallas y transiciones cuando se cambia de una a otra. |

**Tarea:** Cree una aplicación que contenga un formulario de carga de datos, un botón de envío de datos y un texto descriptivo.

2 - StackLayout

StackLayout organiza a sub widgets de forma vertical u horizontalmente, tantos como pueda caber en el ancho o el alto.

Las orientaciones validas desde la última versión considerando top (t) = arriba, right (r) = derecha, bottom (b) = abajo y left (l) = izquierda son:  ‘lr-tb’, ‘tb-lr’, ‘rl-tb’, ‘tb-rl’, ‘lr-bt’, ‘bt-lr’, ‘rl-bt’ y ‘bt-rl’.

|  |  |
| --- | --- |
| **stack/main.py** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **-----------**  **from kivy.uix.stacklayout import StackLayout**    class MiWidget(**StackLayout**):      pass    **-----------** |

|  |  |
| --- | --- |
| **stack/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21** | #:import C kivy.utils.get\_color\_from\_hex  <Button>:      color: .8,.9,0,1      font\_size: 32    <MiWidget>:        orientation: 'rl-tb'      padding: 10      Button:          text: "Link 1..."          size\_hint: [.7, .2]      Button:          text: "Link 2..."          size\_hint: [.3, .3]      Button:          text: "Link 3..."          size\_hint: [.3, .2]      Button:          text: "Link 4..."          size\_hint: [.3, .3] |

A screenshot of a computer

Description automatically generated with medium confidence

3 - PageLayout

La clase PageLayout se usa para crear un diseño simple de varias páginas, de una manera que permite pasar fácilmente de una página a otra.

PageLayout actualmente no respeta las propiedades size\_hint, size\_hint\_min, size\_hint\_max o pos\_hint.

|  |  |
| --- | --- |
| **page/main.py** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **-----------**  **from kivy.uix.pagelayout import PageLayout**    **class MiWidget(PageLayout):**      pass    **-----------** |

|  |  |
| --- | --- |
| **page/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13** | #:import C kivy.utils.get\_color\_from\_hex  <Button>:      color: .8,.9,0,1      font\_size: 32    <MiWidget>:        Button:          text: 'page1'      Button:          text: 'page2'      Button:          text: 'page3' |

Para pasar de una página a otra, nos debemos parar en el borde derecho y arrastrar hacia la izquierda, Para ir hacia atrás nos paramos en el margen izquierdo y arrastramos hacia la derecha.

A screenshot of a computer

Description automatically generated with medium confidence       A screenshot of a computer

Description automatically generated with medium confidence

4 - FloatLayout

Este layout nos permite organizarnos de forma similar a trabajar con la clase Widget principal, salvo que ahora debemos trabajar con coordenadas proporcionales al tamaño total de la ventana en lugar de coordenadas absolutas.

Las coordenadas las asignamos utilizando dos propiedades, nuevas “**size\_hint**” y “**pos\_hint**”, para trabajar de forma proporcional, las cuales toman valores entre 0 y 1. Para ejemplificarlo crearemos un ejercicio con un par de labels y un botón siguiendo el esquema de los ejercicios que venimos manejando pero en donde en lugar de que nuestro widget personalizado herede de la clase “**Widget**”, lo haremos heredar de la clase “**FloatLayout**”.

|  |  |
| --- | --- |
| **float/main.py** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | from kivy.app import App  from kivy.lang import Builder  **from kivy.uix.floatlayout import FloatLayout**    class MiWidget(**FloatLayout**):      pass    class MiApp(App):      def build(self):          return MiWidget()    if \_\_name\_\_ == '\_\_main\_\_':      MiApp().run() |

El siguiente cuadro recoge los valores de los atributos que podemos utilizar con este layout y su aplicación.

|  |  |
| --- | --- |
| **Propiedad** | **Valor** |
| **size\_hint** | Son un par de valores [ancho, altura] en donde ancho y altura expresan un valor proporcional que va de cero (**0**) a uno (**1**) pero también puede ser **None**. |
| **size\_hint\_x**  **size\_hint\_y** | Un valor proporcional que va de cero (**0**) a uno (**1**) pero también puede ser **None**. |
| **pos\_hint** | Es un diccionario en donde el eje “x” viene representado por las claves**: x, center\_x** o **right**) y el eje “y” por las claves: **y, center\_y** o **top**). Las claves toman valores proporcionales al widget entre 0 y 1. |

**Nota**: Los valores son relativos al tamaño de la ventana de la aplicación.

En el siguiente código notemos que no solo podemos heredar atributos compartidos a partir de extender de un widget de kivy como en el caso de los labels, sino que podemos también agregar atributos compartidos a todos los tipo de widgets de kivy existentes como en el caso del botón.

|  |  |
| --- | --- |
| **float/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37** | #:import C kivy.utils.get\_color\_from\_hex  **<MiLabel@Label>:**  **font\_size: 30**  **size\_hint: .3, .3**  **color: 0,0,1,1**  **canvas.before:**  **Color:**  **rgba:  .6,1,.2,1**    **Rectangle:**  **size: self.size**  **pos: self.pos**    **<Button>:**  **color: .8,.9,0,1**  **font\_size: 32**  **size\_hint: .3, .3**    <MiWidget>:        MiLabel:          text: "Pera\n{'x': 0, 'top': 1}"          pos\_hint: {'x': 0, 'top': 1}        MiLabel:            text: "Manzana\n{'right': 1, 'y': 0}"          pos\_hint: {'right': 1, 'y': 0}        Button:          text: "Link 1...\n{'x': 0, 'y': 0}"          pos\_hint: {'x': 0, 'y': 0}        Button:          text: "Link 2...\n{'center\_x':.5, 'center\_y':.5}"          pos\_hint: {'center\_x':.5, 'center\_y':.5}          size\_hint: .6, .3 |

Al ejecutarlo obtenemos:

**A screenshot of a computer screen

Description automatically generated with low confidence**

Si queremos usar las propiedades de tamaño (**size**), altura (**height**) o ancho (**width**), debemos asignar un valor **None** al eje de **size\_hint** que queremos usar con los valores absolutos o si lo hemos indicado por separado a **size\_hint\_x** o **size\_int\_y**. Por ejemplo, tomemos el label que tiene el nombre de manzana y transformémoslo en un label común con el siguiente código:

|  |  |
| --- | --- |
| **float2/mi.kv** | |
| **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37**  **38**  **39** | Label:            text: 'Manzana'  **size\_hint: None, None**  **size: 400, 300**          font\_size: 32          color: 0,0,0,1          pos\_hint: {'x': .7, 'top': .5}          canvas.before:              Color:                  rgba:  .6,.9,.3,1                Rectangle:                  size: self.size                  pos: self.pos |

Aquí el Label está tomando ancho y alto absoluto en “px”

**A screenshot of a computer

Description automatically generated with medium confidence**

También es posible dejar uno de los valores por porcentaje y el otro en “px”, para hacer que tome una altura del 20% de la ventana y de 400px de ancho podemos indicarlo así:

**size\_hint: None, .2**

**width: 400**

**A screenshot of a computer

Description automatically generated**

Nota muy importante:

**Si estamos usando una instancia de un Layout, es posible forzar el uso de valores fijos,  pero puede haber conflictos si no tenemos cuidado con las propiedades que usamos.**

* Si usamos un Layout la prioridad siempre la tienen , pos\_hint y size\_hint.
* Si queremos usar propiedades de posicionamiento fijo (pos, x, center\_x, right, y, center\_y y top), tenemos que asegurarnos de que no estamos usando la propiedad pos\_hint.

5 - RelativeLayout

Este Layout nos permite establecer coordenadas relativas en los elementos hijos. Es decir que a diferencia de “FloatLayout” la posición de un elemento hijo es con relación al “RelativeLayout” en sí, y no con relación a la ventana de la app.

Si creamos un primer ejemplo muy simple, similar al de FloatLayout, podemos ver que parecen responder de forma análoga, ya que salvo la importación de la clase **“RelativeLayout”** de la cual estamos heredando, en ambos estamos utilizando **pos\_int** y **size\_int** para posicionar y dar dimensiones a los botones que lo componen.

|  |  |
| --- | --- |
| **relative1/mi.py** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **-----------**  **from kivy.uix.relativelayout import RelativeLayout**    class MiWidget(**RelativeLayout**):      pass    **-----------** |

|  |  |
| --- | --- |
| **relative1/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17** | #:kivy 2.1.0    #:import C kivy.utils.get\_color\_from\_hex  <Button>:      color: .8,.9,0,1      font\_size: 32      size\_hint: .3, .3    <MiWidget>:        Button:          text: 'Link 1...'          pos\_hint: {'x': 0, 'y': 0}        Button:          text: 'Link 2...'          pos\_hint: {'center\_x':.5, 'center\_y':.5} |

Nos retorna:

**A screenshot of a computer

Description automatically generated with medium confidence**

Esta similitud, nos puede llevar a confusión cuando ponemos un layout dentro de otro para formar una pantalla compleja. Como todas las posiciones dentro de un **RelativeLayout** son relativas a la posición del layout en sí, la posición del layout nunca debe usarse para determinar la posición de los sub-widgets o del canvas del layout. Hay que prestar mucha atención a esto y detenerse en este punto hasta comprenderlo si no queremos aprender algo mal que luego nos dificulte el diseño de la app.

Veamos un ejemplo en el cual vamos a poner un **“RelativeLayout”** y un **“Widget”** dentro de un **“FloatLayout”,** a estas alturas sabemos algunas cosas:

1. El FloatLayout toma sus medidas con relación a la ventana de la app, mientras que el RelativeLayout lo hace con relación a sí mismo.
2. Widget utiliza medidas y posiciones absolutas.
3. Si queremos utilizar medidas absolutas, debemos darle los valores de “None” a size\_hint

El resultado de ejecutar este ejemplo al principio nos puede confundir un poco, por lo que se propone:

1. Leerlo
2. Interpretar que es lo que se esperaría que se vea en pantalla
3. Ejecutarlo
4. Preguntarnos ¿Qué está pasando?
5. Mirar la explicación a continuación de los códigos y de la imagen retornada.

|  |  |
| --- | --- |
| **relative2/mi.py** | |
| **1**  **2**  **3**  **4**  **5**  **6** | **-----------**  **from kivy.uix.floatlayout import FloatLayout**    class MiWidget(**FloatLayout**):      pass  **-----------** |

|  |  |
| --- | --- |
| **relative2/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35** | #:kivy 2.1.0    #:import C kivy.utils.get\_color\_from\_hex    <MiWidget>:      Widget:          size\_hint: None, None          size: 200, 200          pos: 200, 200            canvas:              Color:                  rgba: 1, 1, 1, 1              Rectangle:                  pos: self.pos                  size: self.size        RelativeLayout:          size\_hint: None, None          size: 200, 200          pos: 200, 200            canvas:              Color:                  rgba: .6, 1, .7, .7              Rectangle:                  pos: self.pos  # incorrecto                  size: self.size          Button:              text: "Link 1...\npos\_hint: \n{'x':0, 'y': -1}"              pos\_hint: {'x':0, 'y': -1}          Button:              text: "Link 2...\npos\_hint:\n(self.parent.x, \nself.parent.y - 200)"              #pos: self.parent.pos              pos:(self.parent.x, self.parent.y - 200) |

6 - AnchorLayout

Este layout se utiliza para posicionar otros layouts en lugares específicos, como por ejemplo poner un menú en la parte superior de la pantalla de un dispositivo móvil. No es un layout con el cual podamos maquetar una aplicación completa, sino que su rol es auxiliar.

Todos los elementos dentro de este layout se ubican en la misma posición, mediante los atributos **anchor\_x** y **anchor\_y** con relación a la ventana de la app:

|  |  |
| --- | --- |
| **Propiedad** | **Valor** |
| **anchor\_x** | Toma los valores: ‘left’, ‘center’, ‘right’ |
| **anchor\_y** | Toma los valores: ‘bottom’, ‘center’, ‘top’ |

En el siguiente ejemplo agregaremos tenemos un “AnchorLayout” ocupando el 30% del alto de una ventana, comenzando a posicionarse desde la parte inferior, el cual dentro posee dos botones. Dado que todos los elementos dentro del “AnchorLayout” se van a ubicar en la misma posición, en este caso el margen superior izquierdo del “AnchorLayout”, el segundo botón se hizo más chico para que no oculte totalmente al primero.

|  |  |
| --- | --- |
| **anchor/main.py** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | from kivy.app import App  from kivy.lang import Builder  **from kivy.uix.anchorlayout import AnchorLayout**    class MiWidget(**AnchorLayout**):      pass    class MiApp(App):      def build(self):          return MiWidget()    if \_\_name\_\_ == '\_\_main\_\_':      MiApp().run() |

|  |  |
| --- | --- |
| **anchor/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25** | #:import C kivy.utils.get\_color\_from\_hex    <MiWidget>:      size\_hint: 1, .3  **anchor\_x: 'left'**  **anchor\_y: 'top'**      canvas.before:          Color:              rgba:  .7,.1,.2,1            Rectangle:              size: self.size              pos: self.pos        Button:          color: .8,.9,0,1          font\_size: 32          size\_hint: .3, .3          text: 'Link 1...'        Button:          color: .8,.9,0,1          font\_size: 15          size\_hint: .15, .15          text: 'Link 2...' |

Visualmente nos queda:

**A picture containing screenshot, software, display, multimedia software

Description automatically generated**

**6 - Actionbar**

El widget ActionBar es como el ActionBar de Android, donde los elementos se apilan horizontalmente. Cuando el área se vuelve pequeña, los widgets se mueven al área de ActionOverflow.

Un ActionBar contiene un ActionView con elementos como:

* ActionPrevious con propiedades de title, app\_icon y previous\_icon.
* Subclases de ActionItem como, ActionButton, un ActionToggleButton, un ActionCheck, un ActionSeparator y un ActionGroup. Un ActionGroup se usa para mostrar ActionItems en un grupo y es presentado luego de los  ActionItems.
* Un  ActionOverflow, que se hace visible cuando es necesario, es decir, el área disponible es demasiado pequeña para que quepan todos los widgets.

Un ejemplo al estilo android podría ser el siguiente.

|  |  |
| --- | --- |
| **actionbar/main.py** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | from kivy.app import App  from kivy.lang import Builder  from kivy.uix.widget import Widget  from kivy.uix.gridlayout import GridLayout    class MiWidget(GridLayout):      pass    class MiApp(App):      def build(self):          return MiWidget()    if \_\_name\_\_ == '\_\_main\_\_':      MiApp().run() |

|  |  |
| --- | --- |
| **actionbar/mi.kv** | |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33** | #:import C kivy.utils.get\_color\_from\_hex  <MiWidget>:      cols:2        padding: 15        ActionBar:          pos\_hint: {'top':1}          ActionView:              use\_separator: True              ActionPrevious:                  title: 'Action Bar'                  with\_previous: False              ActionOverflow:              ActionButton:                  icon: 'atlas://data/images/defaulttheme/audio-volume-high'              ActionButton:                  important: True                  text: 'Important'              ActionButton:                  text: 'Btn2'              ActionButton:                  text: 'Btn3'              ActionButton:                  text: 'Btn4'              ActionGroup:                  text: 'Group1'                  ActionButton:                      text: 'Btn5'                  ActionButton:                      text: 'Btn6'                  ActionButton:                      text: 'Btn7' |

A screenshot of a computer

Description automatically generated