tutorial/start.md - Grip

# Animator

Si tratta di disegnare un semplice e simpatico personaggio *Cubetto* e muoverlo per lo schermo del telefonino con dei semplici tap. Ogni volta che tocchiamo lo schermo cubetto raggiungerà il punto toccato girando e cambiando dimensione.

## Rissumiamo

1. Cubetto è un semplice disegno geometrico
2. Quando tocchiamo lo schermo cubetto raggiunge il punto toccato
3. Cobetto si muove rotolando
4. Se tocchiamo la zona sinistra dello schermo cubetto diventa più grande, se tocchiamo la zona destra diventa più piccolo.

## Prima di iniziare

Usate Compilare e Installare per provare il gioco vuoto e come metterlo sul vostro telefonino. Facciamo questo prima di iniziare.

Il programma vuoto contiene il file `main.py` così:

```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout


class Animator(FloatLayout):
    pass


class AnimatorApp(App):
    def build(self):
        return Animator()


if __name__ == "__main__":
    AnimatorApp().run()
```

e il file `animator.kv`

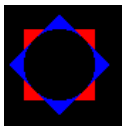
```
#:kivy 1.0.9

<Animator>:
```

## Percorso

1. Cubetto
2. Muoviamo Cubetto
3. Cubetto Cambia Dimesione
4. Cubetto Rotola
5. Contest

tutorial/cubetto.md - Grip

# Cubetto



Cubetto viene disegnato con due quadrati (uno rosso e uno blu) e un cerchio nero al centro per fare il buco.

## Un quadrato bianco

Partiamo con fare un quadrato bianco... quindi aggiungiamo cubetto a animator.

Aggiungiamo il componente `Cubetto` a `Animator` nel fle `animator.kv`

```
#:kivy 1.0.9

<Cubetto>:


<Animator>:
    Cubetto:
        center: root.width*0.5, root.height * 0.5
        size_hint: None,None
        width: dp(50)
```

Abbiamo detto di aggiungere a `<Animator>` un nuovo oggetto cubetto al centro (metà larghezza e metà altezza) largo 50 *punti*.

Eseguiamo e...

```
    File "/usr/local/lib/python2.7/dist-packages/kivy/factory.py", line 131, in __getattr__
      raise FactoryException('Unknown class <%s>' % name)
  kivy.factory.FactoryException: Unknown class <Cubetto>
```

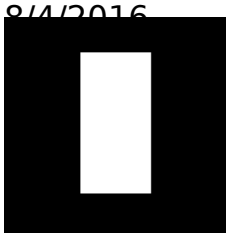
... Manca Cubetto in `main.py` . Aggiungiamo quindi come un semplice `Widget` in `main.py`

```
from kivy.uix.widget import Widget


class Cubetto(Widget):
    pass
```

Riproviamo... ora funziona me è ancora tutto nero: giusto ... ci siamo dimenticati di disegnare il quadrato. Dentro `animator.kv` modifichiamo `<Cubetto>` come

```
<Cubetto>:
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size
```

Proviamo e ...



Non è proprio quello che volevamo.... l'altezza deve essere uguale alla larghezza! Diciamolo aggiungendo la riga `height: self.width` a `<Cubetto>` .

```
<Cubetto>:
    height: self.width
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size
```

Ecco il nostro quadrato bianco.

## Il quadrato diventa Rosso

Per far diventare il quadrato rosso bisogna dire semplicemente di usare il colore rosso prima di disegnare il quadrato.

Per indicare il colore rosso basta usare `rgb: (1,0,0)` e quindi modificare `<Cubetto>` prima di disegnare il rettangolo

```
    Color:
        ....
```

Esistono però tanti modi per indicare un colore. Noi useremo

```
    rgb: (rosso,verde,blu)
```

Dove `rosso` , `verde` e `blu` sono tre valori tra 0 e 1 che indicano quanto di quel colore usare. Quindi il rosso diventa `rgb: (1,0,0)`

```
<Cubetto>:
    height: self.width
    canvas:
        Color:
            rgb: (1,0,0)
        Rectangle:
            ....
```

Proviamo e ... **ecco il quadrato rosso**.

Prima di andare avanti costruiamoci un po di colori da usare invece dei numeri: all'inizio di `animator.kv` aggiungiamo i nostri colori e sostuiamo `(1,0,0)` con rosso.

```
#:kivy 1.0.9

#:set nero (0, 0, 0)
#:set bianco (1, 1, 1)
#:set rosso (1, 0, 0)
#:set verde (0, 1, 0)
#:set blu (0, 0, 1)


<Cubetto>:
    height: self.width
    canvas:
        Color:
            rgb: rosso
        Rectangle:
            ....
```

## Il quadrato blu

Ora dobbiamo fare un quadrato blu.... ma girato di 45 gradi. Per girare bisogna indicare di fare una rotazione del *foglio* prima di disegnare il prossimo rettangolo. Quindi:

1. Ruotare il foglio
2. Colore
3. Rettangolo

Il 2 e il 3 li sappiamo già fare, ma per ruotare?

Con

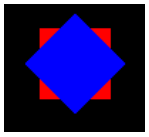
```
        Rotate:
            angle: 45
            origin: self.center
```

dentro alla **tela** ( `canvas` ) si dice di ruorare la **tela** di 45 gradi tenendo il centro fisso prima di fare i prossimi disegni.

Quindi dopo aver disegnato il rettangolo rosso aggiungiamo:

```
<Cubetto>:
    height: self.width
    canvas:
        .... Il rettangolo rosso
        Rotate:
            angle: 45
            origin: self.center
        Color:
            rgb: blu
        Rectangle:
            pos: self.pos
            size: self.size
```

Ecco cosa viene fuori:



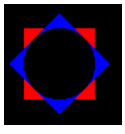
## Il cerchio nero al centro

Per fare l'effetto *buco* basta fare un cerchio dentro ai quadrati. I cerchi pieni si costruiscono usando la parola `Ellipse` che deve avere la stassa posizione e dimensione dei `Rectangle` ... ricordatevi di usare il colore `nero` .

Quindi dopo i rettangoli aggiungiamo:

```
<Cubetto>:
    ...
    canvas:
        Color:
            rgb: nero
        Ellipse:
            pos: self.pos
            size: self.size
```

**BELLO!**



- **NEXT** Muoviamo Cubetto

- **PREV** Indice
- **INDEX** Indice

tutorial/muovi.md - Grip

# Muoviamo Cubetto

Per muovere cubetto usiamo il touch, quando tocchiamo `on_touch_down(touch)` spostiamo il centro di cubetto nel punto dove abbiamo toccato.

1. Prima capiamo dove scrivere le cose
2. Spostiamo cubetto con il *teletrasporto*
3. Facciamo arrivare cubetto a destinazione *dolcemente*

## Dove scrivere le cose

Noi vogliamo *catturare* il punto dove tocchiamo lo schermo. Sul nostro schermo abbiamo messo il componente `Animator`, quindi chiediamo a lui quando viene toccato: in `main.py` nella classe `Animator` aggiungiamo

```
class Animator(FloatLayout):
    def on_touch_down(self, touch):
        print(touch.pos)
```

Ora proviamo, ogni volta che tappiamo (o clickiamo) vengono stampate le coordinate del punto dove clickiamo che sono contenute in `touch.pos`:

```
(181.0, 357.0)
(554.0, 386.0)
(539.0, 157.99999999999997)
(318.0, 139.99999999999997)
```

## Spostiamo cubetto

Prima di spostare cubetto bisogna fare in maniera che `Animator` conosca cubetto. Per far questo aggiungiamo una proprietà a `Animator` di tipo oggetto e *attachiamoci* cubetto. Nel file `main.py` bisogna aggiungere in alto `from kivy.properties import ObjectProperty` e nella classe `Animator` la nuova property:

```
from kivy.properties import ObjectProperty

...

class Animator(FloatLayout):
    cubetto = ObjectProperty(None)

    ....
```

Ora dobbiamo associare questa property al componente `Cubetto` creato nel file `animator.kv`:

```
<Animator>:
    cubetto: cubetto_id

    Cubetto:
        id: cubetto_id
        ...
```

Nota: alla base di `Animator` abbiamo messo `cubetto` come quello della classe per poi associare quel valore a `id` del `Cubetto` creato dentro `<Animator>`.

Bene ora possiamo spostare `cubetto` dal componente `Animator` semplicemente cambiandogli le sue coordinate o il suo centro: se dentro `on_touch_down()` mettiamo `self.cubetto.center = touch.pos` spostiamo cubetto nel punto toccato (`touch.pos`):

```
class Animator(FloatLayout):
    cubetto = ObjectProperty(None)

    def on_touch_down(self, touch):
        self.cubetto.center = touch.pos
```

## Animazione morbida

Invece di usare il teletrasporto vogliamo che cubetto arrivi morbidamente nella nuova posizione. Per fare quasto `kivy` mette a disposizione uno strumento bellissimo `Animation`. Per usarlo in alto mettete `from kivy.animation import Animation` e poi modificate `on_touch_down` così:

```
    def on_touch_down(self, touch):
        animation = Animation(center=touch.pos)
        animation.start(self.cubetto)
```

State dicendo di costruire una animazione che sposta il centro in `touch.pos` che dura 1 secondo (se non si indica `duration` la durata è un secondo), che traforma *linearmente* (se non si indica `t`).

E' possibile cambiare più proprietà contemporaneamente (lo vediamo dopo), cambiare la durata e il tipo di trasformazione.

Provate a giocare usando le seguenti modifiche:

- `animation = Animation(center=touch.pos, duration=2)`
- `animation = Animation(center=touch.pos, duration=0.5)`
- `animation = Animation(center=touch.pos, duration=2, t='in_out_quad')`
- `animation = Animation(center=touch.pos, t='out_back')`
- `animation = Animation(center=touch.pos, t='out_bounce')`

tutti i valori possibili di `t` si trovano alla pagina di ducumentazione di kivy.

Mettete come valore adesso:

```
animation = Animation(center=touch.pos, t='in_out_quad')
```


- **NEXT** Cubetto Cambia Dimesione
- **PREV** Cubetto
- **INDEX** Indice

tutorial/dimensione.md - Grip

# Cubetto Cambia Dimesione

Per cambiare la dimensione di cubetto basta cambiare la sua `width` che all'inizio abbiamo fissato a `dp(50)`.

1. Prima proviamo a aumentare la dimesione quando facciamo il tap
2. La modifica la facciamo diventare *morbida*
3. Quando tappiamo a destra la dimensione diminuisce e quando tappiamo a sinistra aumenta

## Aumentiamo la dimensione

Aumentiamo la dimensione di `1.3`. Quindi quando facciamo `on_touch_down()` prima della nostra animazione cambiamo `self.cubetto.width` moltiplicandolo per `1.3`. All'inizio di `on_touch_down()` mettiamo:

```
    def on_touch_down(self, touch):
        larghezza = self.cubetto.width * 1.3
        self.cubetto.width = larghezza
```

e proviamo...

**Bene** ... ora ripuliamo facendo diventare `1.3` un numero facile da modificare e usando una funzione fare il calcolo.

La classe `Animator` diventa:

```
class Animator(FloatLayout):
    cubetto = ObjectProperty(None)
    INCREMENTO_LARGHEZZA = 1.3

    def on_touch_down(self, touch):
        larghezza = self.ingrandisci()
        self.cubetto.width = larghezza
        animation = Animation(center=touch.pos, t='in_out_quad')
        animation.start(self.cubetto)

    def ingrandisci(self):
        return self.cubetto.width * self.INCREMENTO_LARGHEZZA
```

## Cambiamo la dimensione morbidamente

Per cambiare la dimensione *morbidamente* basta cambiare l'animazione `Animator` dicendogli di modificare anche `width` oltre a `center`: `Animation(center=touch.pos, width=larghezza t='in_out_quad')`. Togliamo anche la modifica istantanea che ora non serve più.

```
    def on_touch_down(self, touch):
        larghezza = self.ingrandisci()
        animation = Animation(center=touch.pos, width=larghezza, t='in_out_quad')
        animation.start(self.cubetto)
```

Ora è bello *morbido*.

## Ingrandiamo e Rimpiccioliamo

Prima di far la scelta se ingrandire o rimpicciolire proviamo a cambiare il programma per rimpicciolire. Facciamo una nuova funzione `rimpicciolisci()` che divide la dimesione invece di ingrandire e cambiamo `larghezza` usandola:

```
    def rimpicciolisci(self):
        return self.cubetto.width / self.INCREMENTO_LARGHEZZA
```

e dentro `on_touch_down()` mettiamo: `larghezza = self.rimpicciolisci()` .

Adesso quando la posizione `x` del tocco `touch` è maggiore di metà della larghezza di `Animator` ingrandiamo, altrimenti rimpiccioliamo.... chiaro no. Questa frase si traduce in:

```
        x, y = touch.pos
        if x > self.width/2:
            larghezza = self.rimpicciolisci()
        else:
            larghezza = self.ingrandisci()
```

quindi `on_touch_down()` diventa

```
    def on_touch_down(self, touch):
        x, y = touch.pos
        if x > self.width/2:
            larghezza = self.rimpicciolisci()
        else:
            larghezza = self.ingrandisci()
        animation = Animation(center=touch.pos, width=larghezza, t='in_out_quad')
        animation.start(self.cubetto)
```

Tutto morbido e si muove come vogliamo.


- **NEXT** Cubetto Rotola
- **PREV** Muoviamo Cubetto
- **INDEX** Indice

# Cubetto Rotola

Dobbiamo far rotolare, anzi ruotare su se stesso cubetto. Sappiamo già come fare per *ruotare il foglio di disegno* dove cubetto viene disegnato. Dobbiamo solo fare in maniera di poter scegliere l'angolo senza doverlo scrivere a mano.

## Ruotiamo Cubetto

Quindi se vogliamo ruotare cubetto di 32 gradi dobbiamo *ruotare la tela di 32 gradi prima di disegnare il resto*. Nella tela ( `canvas` ) di `<Cubetto>` facciamo una rotazione di 32 prima di tutto il resto: `animator.kv`

```
<Cubetto>:
    height: self.width
    canvas:
        Rotate:
            angle: 32
            origin: self.center
        ...
```

Adesso cubetto è girato.

## angolo, una nuova proprietà di Cubetto

Vogliamo quindi definire una nuova proprità di `Cubetto` come la larghezza ( `width` ) o l'altezza ( `height` ). In `main.py` cambiamo `Cubetto` aggiungendo la proprietà numerica `angolo` . Mettiamo in alto

```
from kivy.properties import NumericProperty
```

e cambiamo `Cubetto`

```
class Cubetto(Widget):
    angolo = NumericProperty(32)
```

**GRANDE**: Ora possiamo sostituire nel file `animator.kv` il numero 32 con `self.angolo` :

```
<Cubetto>:
    height: self.width
    canvas:
        Rotate:
            angle: self.angolo
            origin: self.center
        ...
```

Ora cambiando `angolo` di cubetto possiamo ruotarlo. Facciamolo a ogni movimento.

## Rotazione.... morbida

Ormai abbiamo capito: basta cambiare `Animation()` aggiungendo `angolo=nuovo_angolo` e `nuovo_angolo` lo calcoliamo sommando `417` al vecchio.

```
        animation = Animation(center=touch.pos, width=larghezza, angolo=self.cubetto.angolo + 417 , t='in_out_quad')
```

Ancora un piccolo sforzo:

1. `417` lo chiamiamo `INCREMENTO_ANGOLO`

2. facciamo una funziona `ruota()` per calcolare il `nuovo_angolo`

```
class Animator(FloatLayout):
    ...
    INCREMENTO_ANGOLO = 417

    def on_touch_down(self, touch):
        ...
        nuovo_angolo = self.ruota()
        animation = Animation(center=touch.pos, width=larghezza, angolo=nuovo_angolo, t='in_out_quad')
        animation.start(self.cubetto)

    def ruota(self):
        return self.cubetto.angolo + self.INCREMENTO_ANGOLO
```

**ECCO FATTO**

`main.py`

```
from kivy.animation import Animation
from kivy.app import App
from kivy.properties import ObjectProperty, NumericProperty
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.widget import Widget


class Cubetto(Widget):
    angolo = NumericProperty(32)


class Animator(FloatLayout):
    cubetto = ObjectProperty(None)
    INCREMENTO_LARGHEZZA = 1.3
    INCREMENTO_ANGOLO = 417

    def on_touch_down(self, touch):
        x, y = touch.pos
        if x > self.width/2:
            larghezza = self.rimpicciolisci()
        else:
            larghezza = self.ingrandisci()
        nuovo_angolo = self.ruota()
        animation = Animation(center=touch.pos, width=larghezza, angolo=nuovo_angolo, t='in_out_quad')
        animation.start(self.cubetto)

    def ruota(self):
        return self.cubetto.angolo + self.INCREMENTO_ANGOLO

    def ingrandisci(self):
        return self.cubetto.width * self.INCREMENTO_LARGHEZZA

    def rimpicciolisci(self):
        return self.cubetto.width / self.INCREMENTO_LARGHEZZA


class AnimatorApp(App):
    def build(self):
        animator = Animator()
        from utils import PrintScreenshoot
        self._print_handler = PrintScreenshoot(animator)
        return animator


if __name__ == "__main__":
    AnimatorApp().run()
```

`animator.kv`

```
#:kivy 1.0.9
#:set nero (0, 0, 0)
#:set bianco (1, 1, 1)
#:set rosso (1, 0, 0)
#:set verde (0, 1, 0)
```

```
#:set blu (0, 0, 1)

<Cubetto>:
    height: self.width
    canvas:
        Rotate:
            angle: self.angolo
            origin: self.center
        Color:
            rgb: rosso
        Rectangle:
            pos: self.pos
            size: self.size
        Rotate:
            angle: 45
            origin: self.center
        Color:
            rgb: blu
        Rectangle:
            pos: self.pos
            size: self.size
        Color:
            rgb: nero
        Ellipse:
            pos: self.pos
            size: self.size


<Animator>:
    cubetto: cubetto_id

    Cubetto:
        id: cubetto_id
        center: root.width*0.5, root.height * 0.5
        size_hint: None,None
        width: dp(50)
```


- **NEXT** Cubetto Contest
- **PREV** Cubetto Cambia Dimesione
- **INDEX** Indice

tutorial/contest.md - Grip

# Contest

Abbiamo imparato tante cose, ora provate da soli a cambiare un po di cose:

- Cambiate il disegno di `Cubetto` ... unico limite, la fantasia
- Cambiate lo sfondo da nero a bianco
- Animazione diversa per spostamento rotolante e dimensione
- Animazione, prima spostamento rotolante e poi dimensione

## Sfondo

Aiutino: Fate un rettangolo bianco in `<Animator>` ....

## Animazione

Aiutino: una volta fatta una animazione `animazione` possiamo mettere una nuova animazione in *parallelo* usando

```
animazione |= Animation(...)
```

e una dopo l'altra con

```
animazione += Animation(...)
```


- **PREV** Cubetto Cubetto Rotola
- **INDEX** Indice