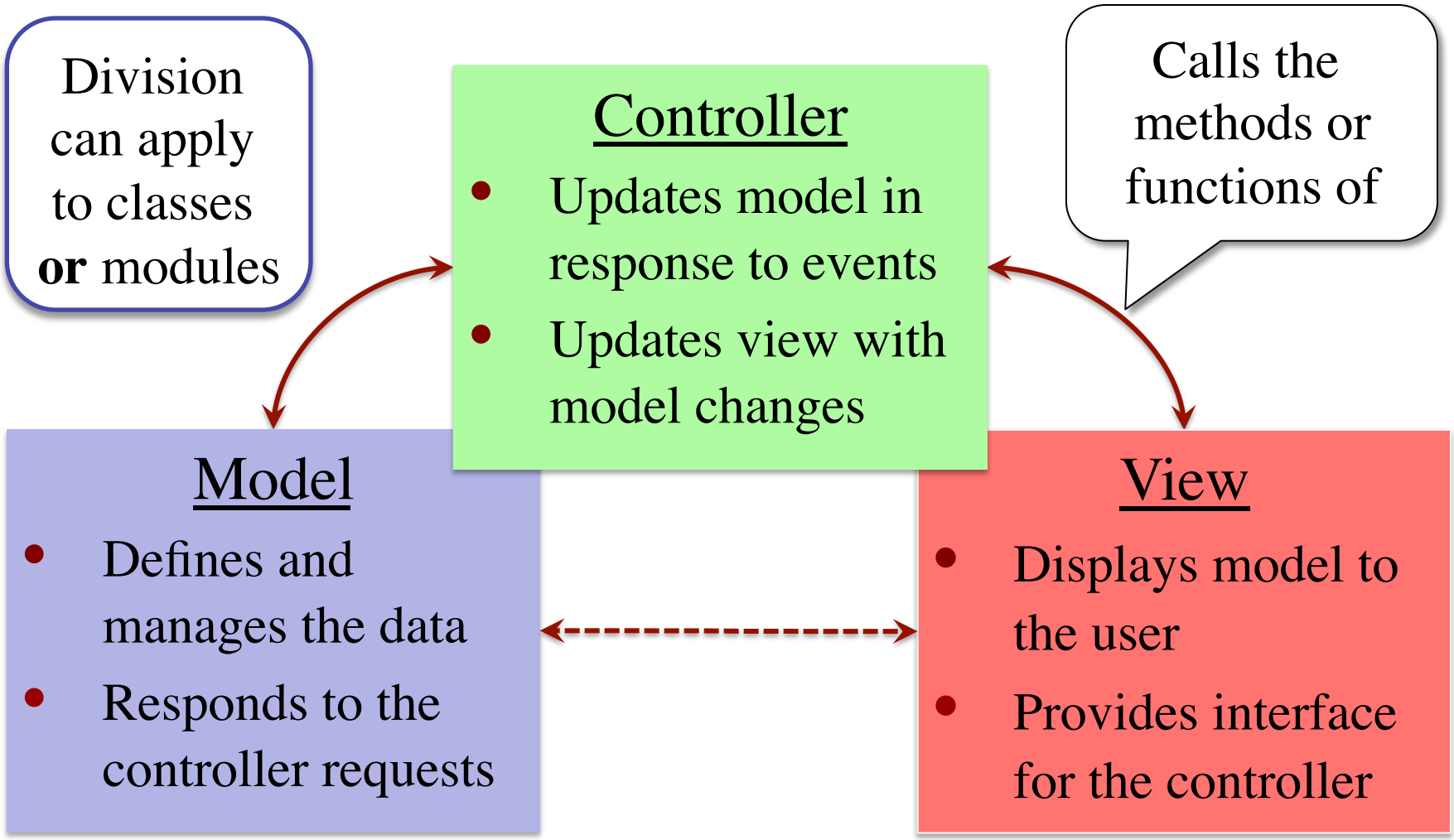
Lecture 24

Callbacks & Stateful Controllers

Model-View-Controller Pattern



MVC in this Course

Model

- A3: Color classes
  - RGB, CMYK & HSV
- A4: Turtle, Pen
  - Window does the drawing
- A5: Matrix, Vector
- A6: ImageArray
- A7: Ball, Paddle, Bricks

Controller

- A3: Functions in a3.py
  - No need for classes
- A4: Functions in a4.py
  - No need for classes
- A5: Nothing you wrote
- A6: ImageProcessor
- A7: Breakout

MVC in this Course

Model

- A3: Color classes
  - RGB, CMYK & HSV
- A4: Turtle, Pen
  - Window does the drawing
- A5: Matrix, Vector
- A6: ImageProcessor
- A7: Ball, Paddle, Bricks

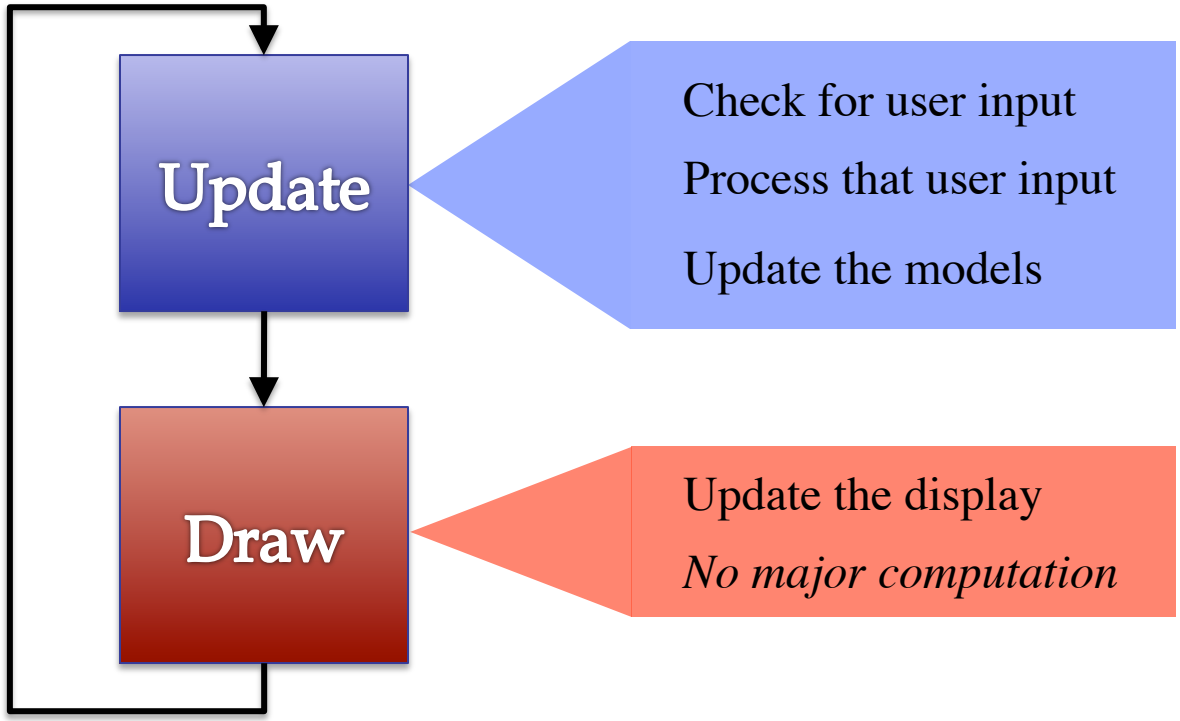
Controller

- A3: Functions in a3.py
  - No need for classes
- A4: Functions in a4.py
  - No need for classes
- A5: Nothing you wrote
- A6: ImageProcessor
- A7: Breakout

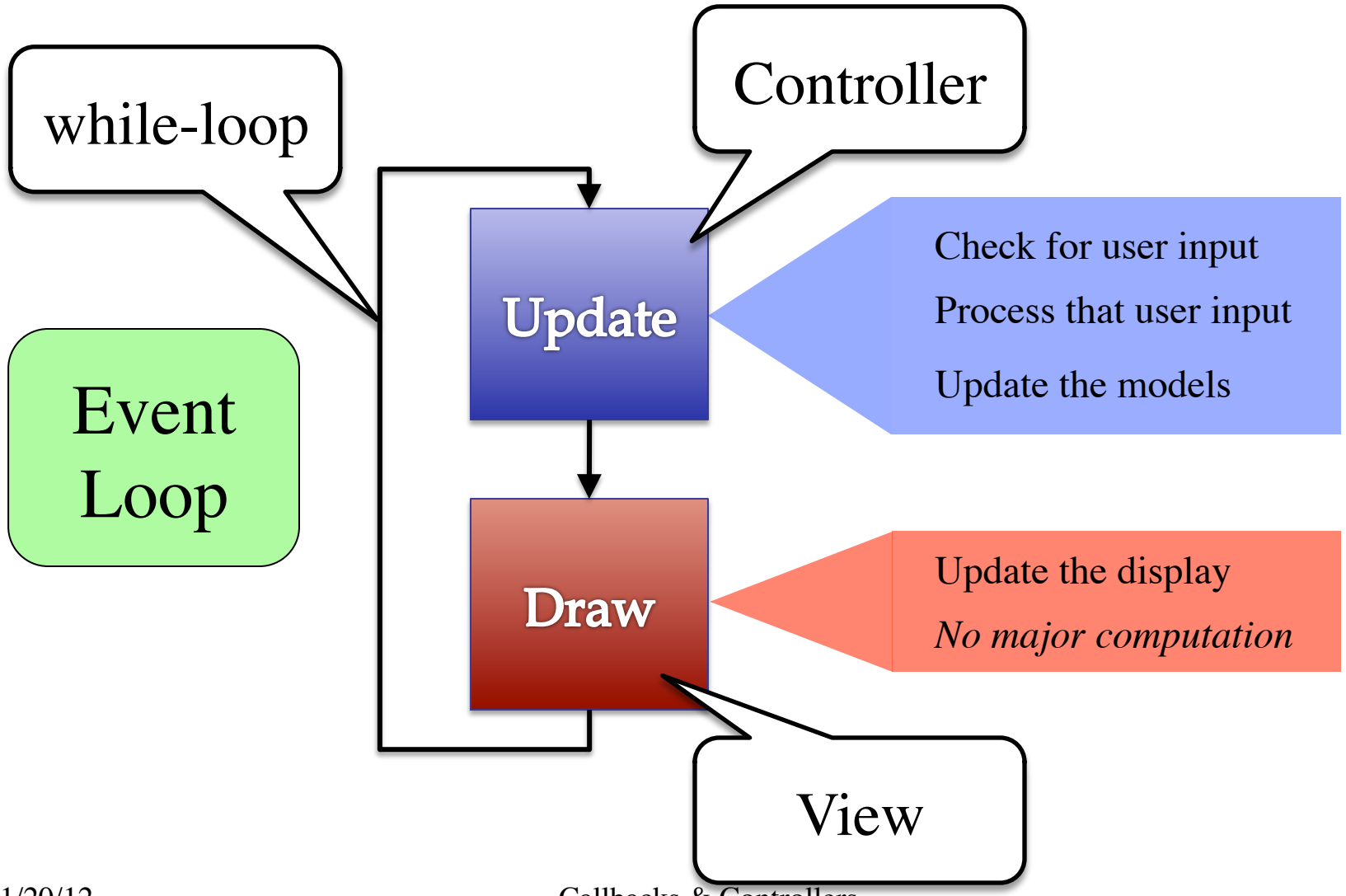
When need functions and when need classes?

A Standard GUI Application

Animates the application, like a movie



A Standard GUI Application



Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Handled by OS/GUI libraries

    # Your code goes here

    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Ha                                  libraries

    # Your code goes here



    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Would like to "plug in" code

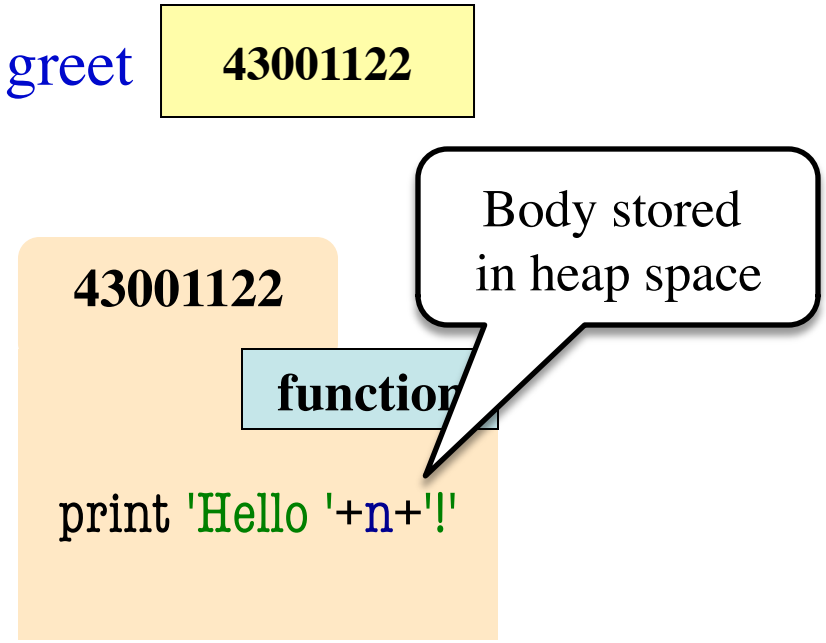
Why do we need to write this each time?

Function Names are Variables

- Calling a function
  - Provide arguments in ()
  - Executes the body
- Passing a function
  - Assign another variable
  - Use the name without ()
- Example:

```
>>> x = greet
>>> x('Walker')
Hello Walker!
```

```
def greet(n):
    print 'Hello '+n+'!'
```



Callback Functions

- **Given**: predefined code that calls some function
  - But function not defined
  - You want to replace it with your function
- Assign that function to the name of yours
  - When called, it *calls back* to your function definition
  - *Sort of* like overriding
  - But can't get old version

```
callback = <your function>
...
while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw
```

See callback.py

Application: Buttons

- Buttons in Kivy all have a special attribute
  - Named on_press
  - Stores a function
- Called on button press
  - Assign it what you want
- Standard for GUI apps
  - Libraries do hard work
  - Customize behavior w/ callback functions

```
class ButtonMain(Widget):
    """Kivy window with a single button"""

    def __init__(self,**kw):
        """Constructor: make panel w/ button"""
        super(ButtonMain,self).__init__(**kw)
        button = Button(text='Click Me!',
                        size_hint=(1,1))
        self.add_widget(button)

        # Set the callback function
        button.on_press = self.my_callback

    def my_callback(self):
        """Function to call on button press."""
        print 'Hello World!'
```

Loop Invariants Revisited

Normal Loops

Properties of "external" vars

```
x = 0
i = 2
# x = sum of squares of 2..i
while i <= 5:
    x = x + i*i
    i = i + 1
# x = sum of squares of 2..5
```

Loops & Callbacks

What are the "external" vars?

```
while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw
```

Loop Invariants Revisited

Normal Loops

```
x = 0
i = 2
# x = sum of squares of 2..i
while i <= 5:
    x = x + i*i
    i = i + 1
# x = sum of squares of 2..5
```

Properties of "external" vars

Loops & Callbacks

What are the "external" vars?

```
while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw
```

If callback a method, then it has attributes

Attribute Invariants = Loop Invariants

- Fields are only way to store value between calls
  - Not part of call frame
  - Variables outside loop
- So all callback functions should be methods
  - Variable stores function definition **and** the object
  - Knows to call method on that particular object
  - Uses its fields for state

```
callback = obj.method
...
# inv: obj attributes are ...
while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw
# post: obj attributes are ...
```

Example: Animation

- Callback: animate(...)
  - Called 60x a second
  - Moves back and forth
- Animate is a method
  - Associated with an object
  - Object has changing state
- **Examples** of state
  - Ellipse position
  - Current velocity
  - Current animation step

```
def animate(self,dt):
    """Animate the ellipse back & forth"""
    if self._steps == 0:
        # Initialize
        ...
    elif self._steps > ANIMATION_STEPS/2:
        # Move away
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x+self._vx,y+self._vy)
        self._steps = self._steps - 1
    else:  # Move back
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x-self._vx,y-self._vy)
        self._steps = self._steps - 1
```

Example: Animation

- Callback: animate(...)
  - Called 60x a second
  - Moves back and forth
- Animate is a method
  - A ect
  - O anging state
- **Examples** of state
  - Ellipse position
  - Current velocity
  - Current animation step

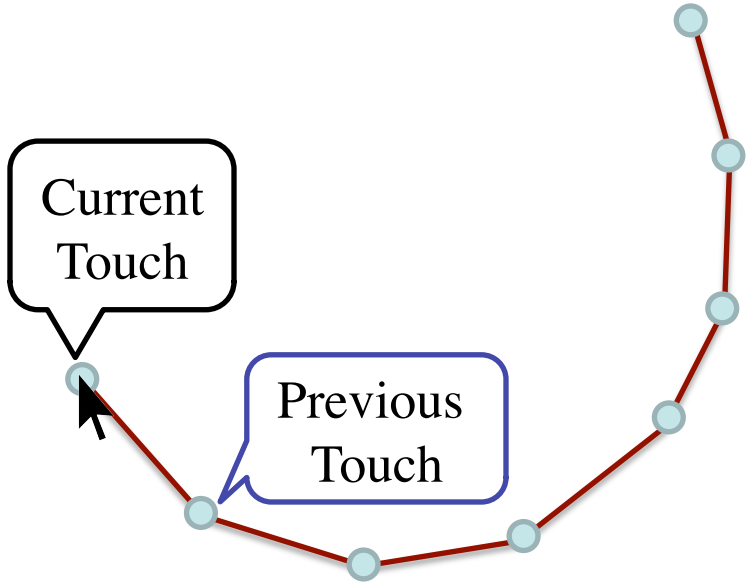
See animate.py

```
def animate(self,dt):
    """Animate the ipse back & forth"""
    if self._steps ==
        # Initia
        ...
    elif self._
        # Move away
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x+self._vx,y+self._vy)
        self._steps = self._steps - 1
    else:  # Move back
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x-self._vx,y-self._vy)
        self._steps = self._steps - 1
```

Kivy requires argument in animation callbacks

State Across Multiple Callbacks

- Sometimes have more than one callback function
- Example: touch events
  - `on_touch_down`:
    User presses mouse (or a finger); does not release
  - `on_touch_up`:
    Releases mouse (or finger)
  - `on_touch_move`:
    Moves mouse (or finger)
- State needed to track change in touch over time



See touch.py

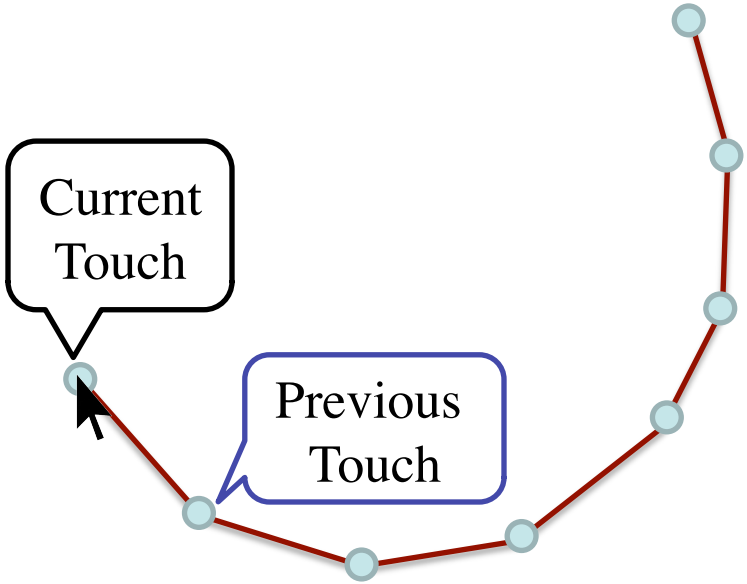
State Across Multiple Callbacks

```
# None or previous touch
_anchor = None

def on_touch_down(self,touch):
    # Track touch state
    self._anchor = (touch.x,touch.y)

 def on_touch_up(self,touch):
    # Nothing to track
    self._anchor = None

 def on_touch_move(self,touch):
    if not self._anchor is None:
        self.drawLine(self._anchor[0], self._anchor[1],
                      touch.x,touch.y,LINE_COLOR)
        self._anchor = (touch.x,touch.y)
```



See touch.py