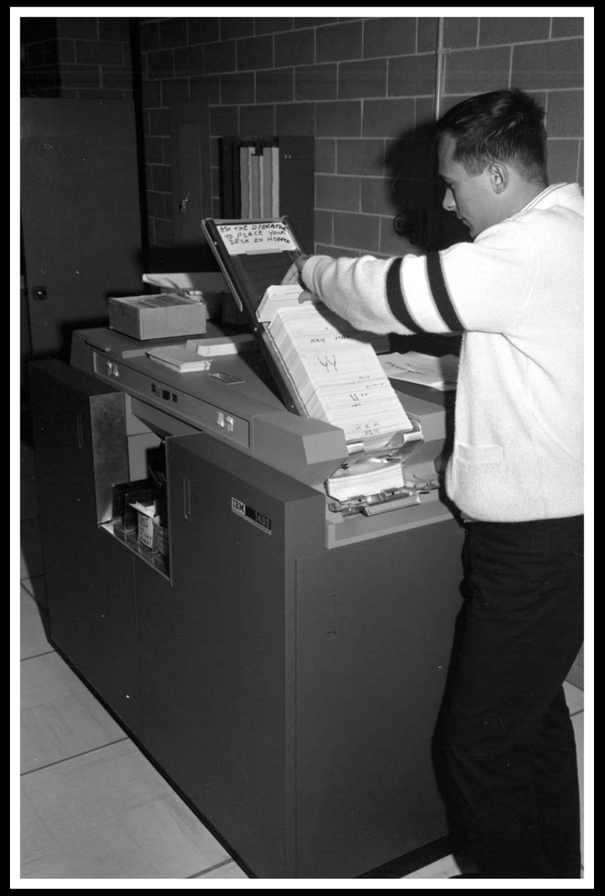
Interaction

AME 230 - Programming for Media Arts

Life cycle of a batch processing, non-interactive program

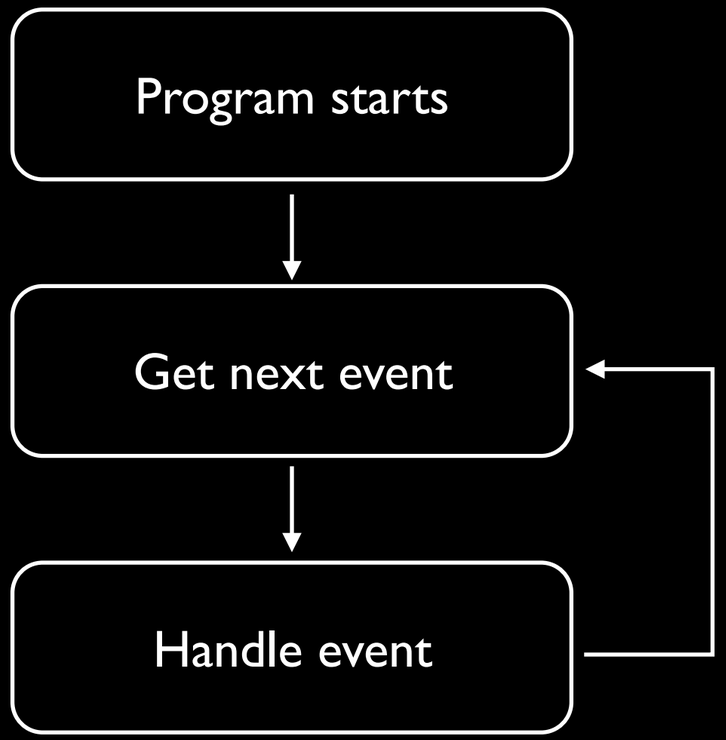


Program starts

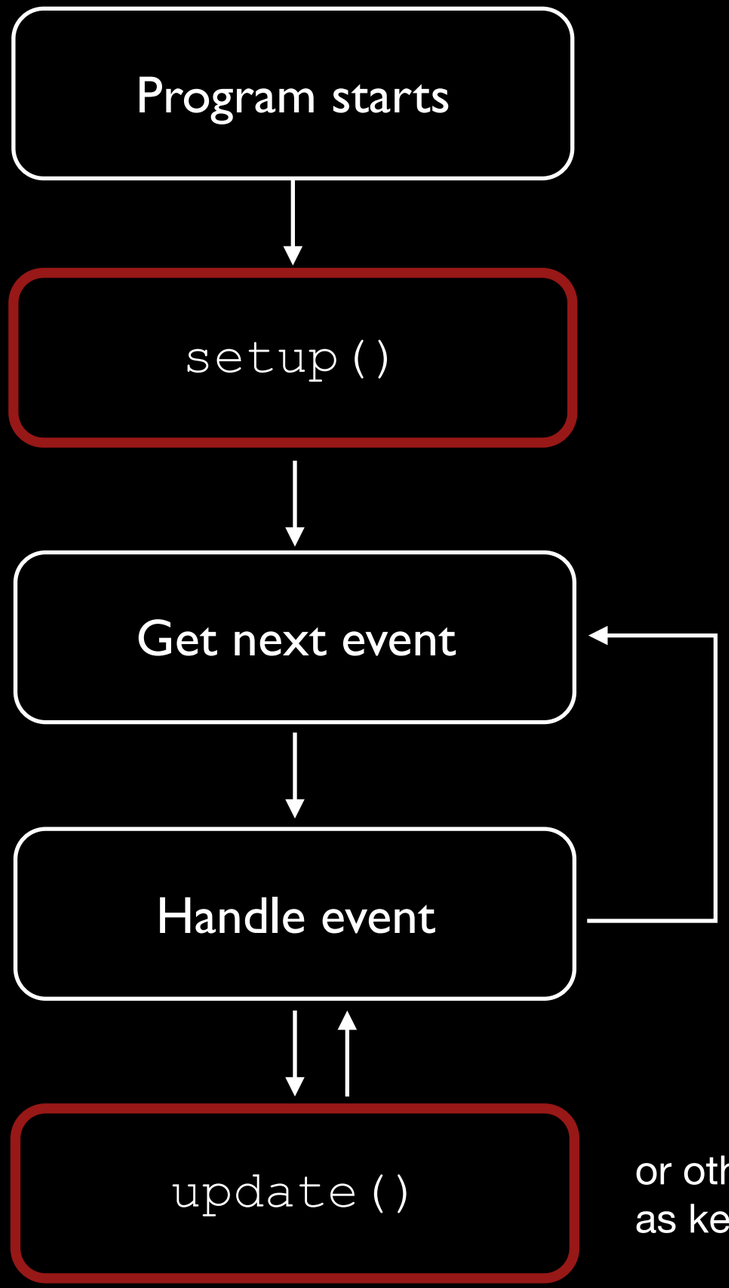
```
instruction 1
instruction 2
instruction 3
      •
      •
      •
instruction N
```

Program ends

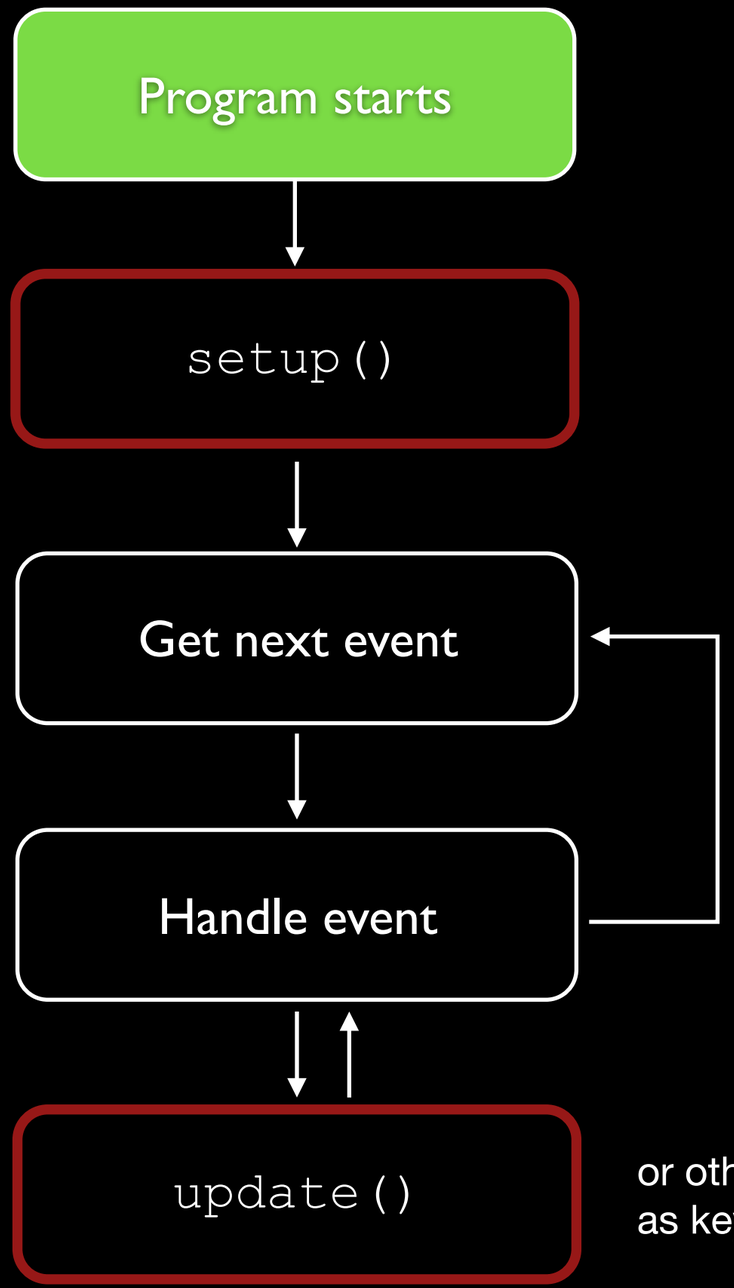
Life cycle of an event-loop (eg. interactive) program



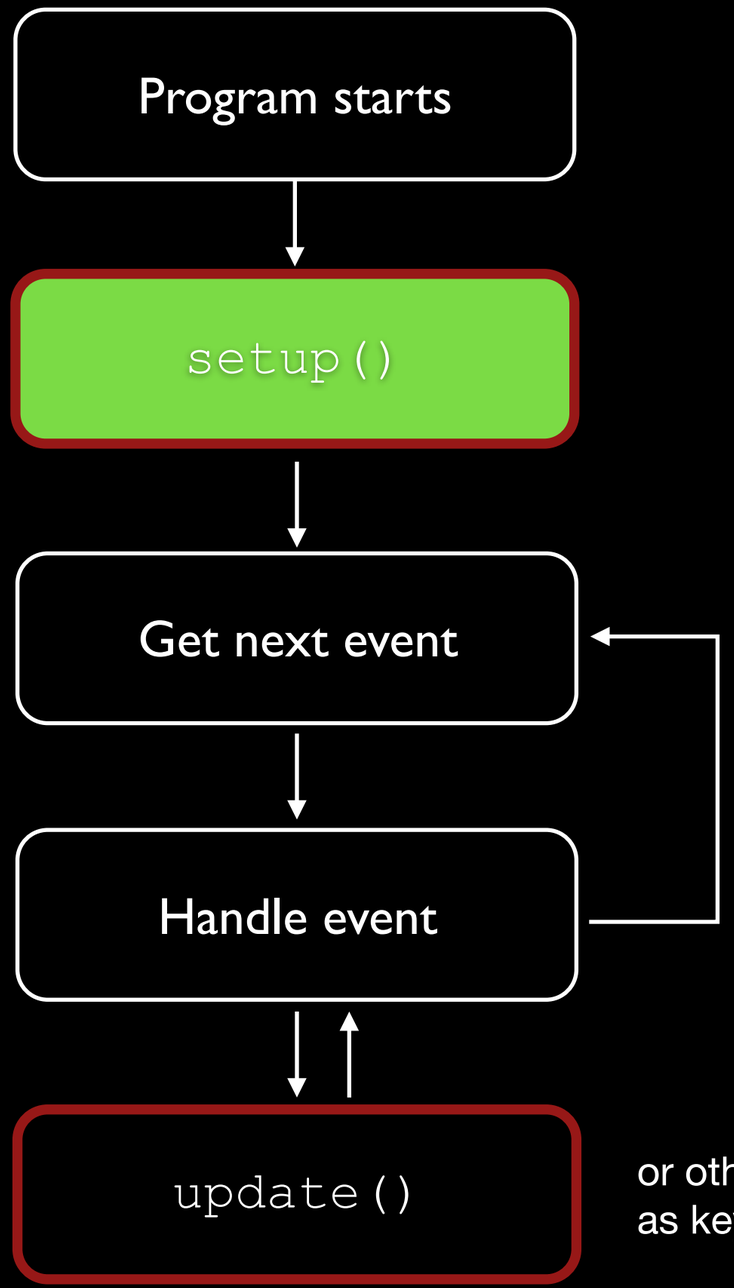
Life cycle details for an interactive program



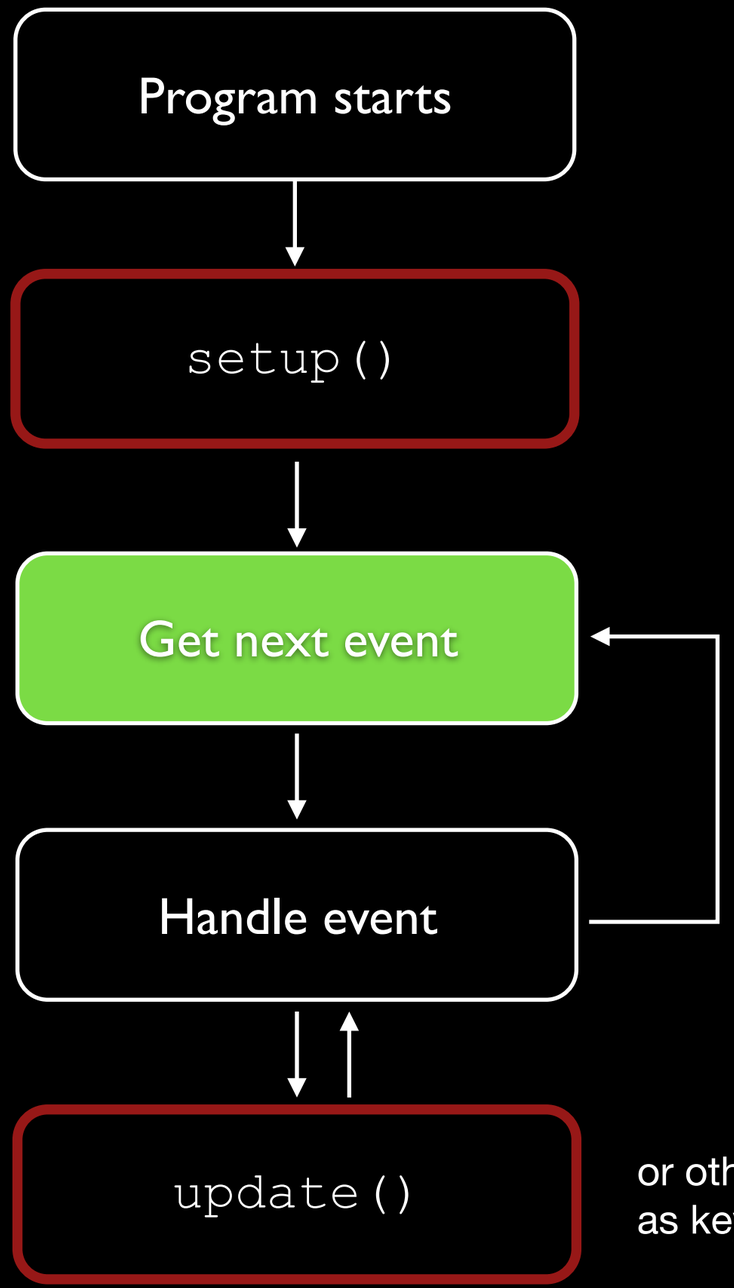
or other event handlers for things such as keyboard or mouse (pointer) events.



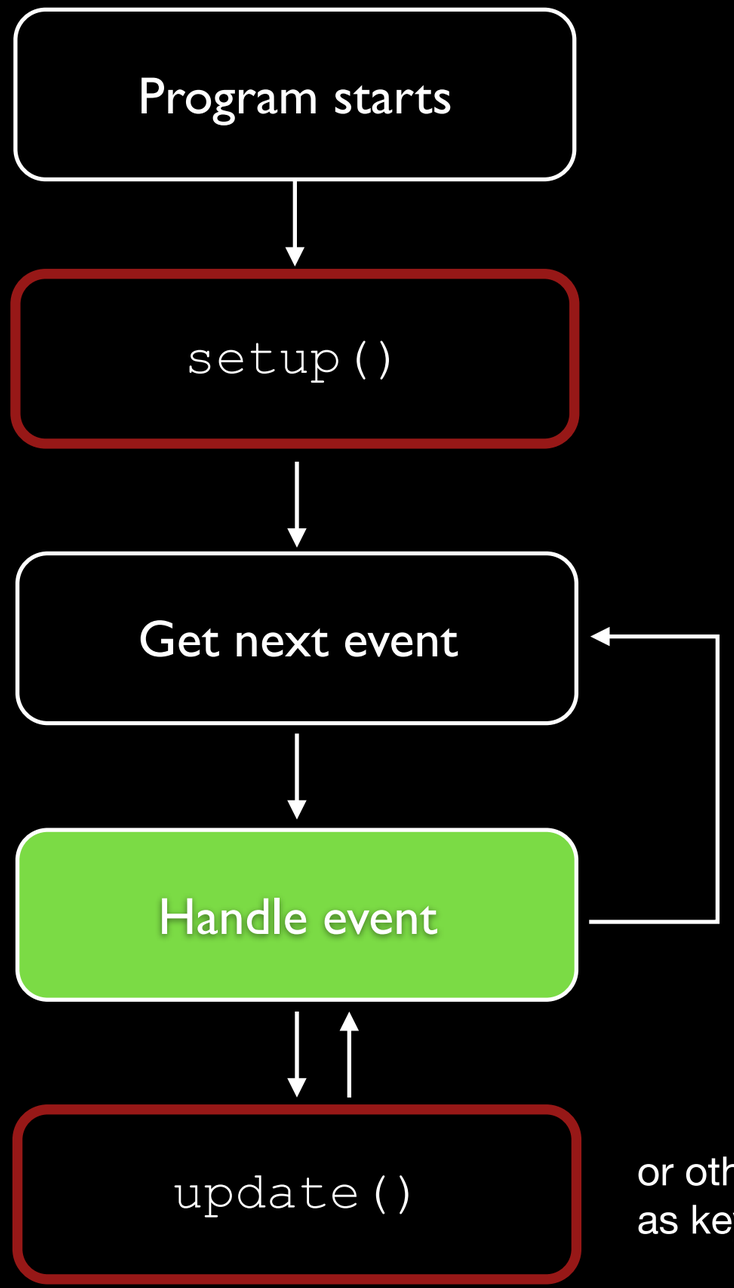
or other event handlers for things such as keyboard or mouse (pointer) events.



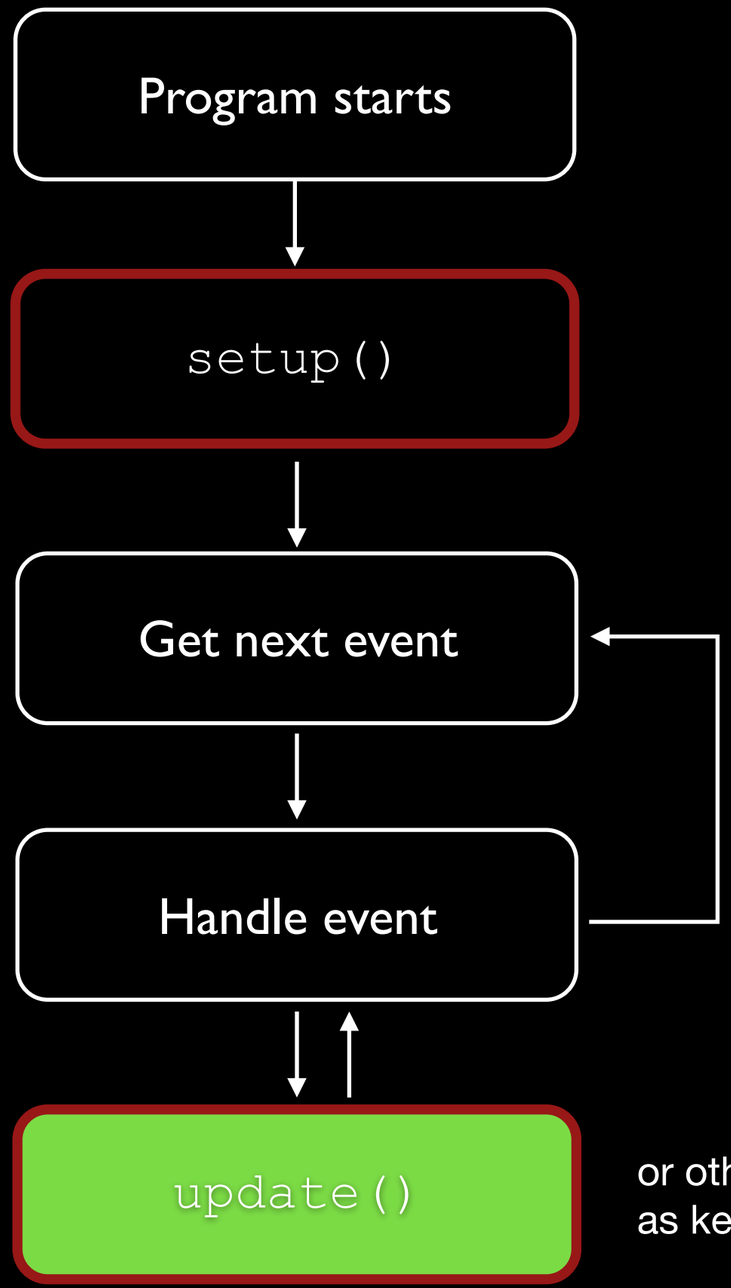
or other event handlers for things such as keyboard or mouse (pointer) events.



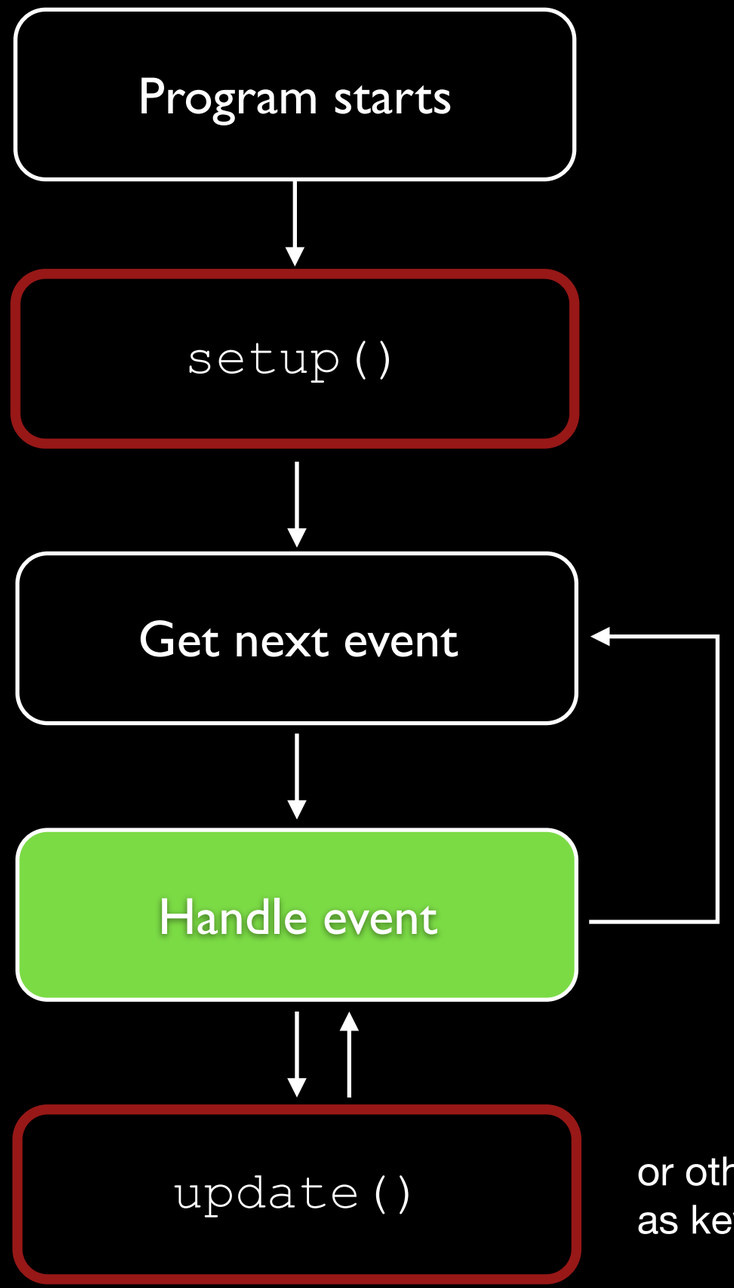
or other event handlers for things such as keyboard or mouse (pointer) events.



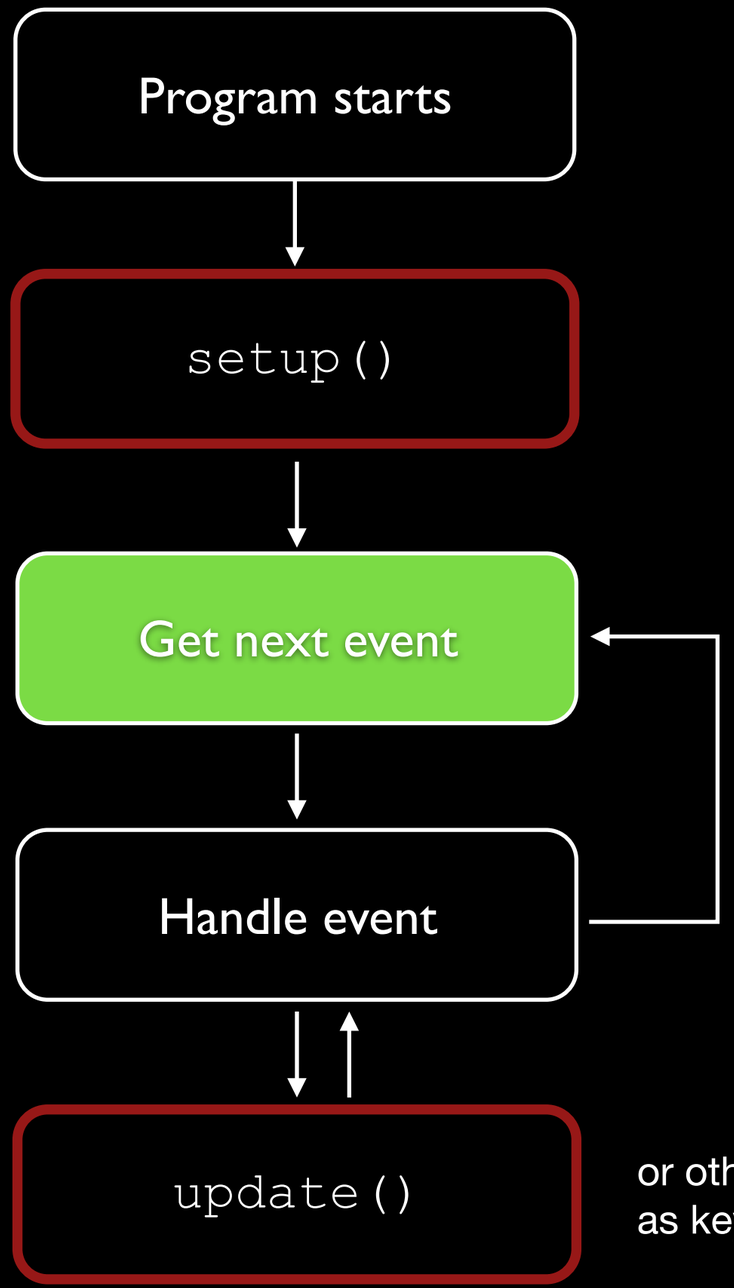
or other event handlers for things such as keyboard or mouse (pointer) events.



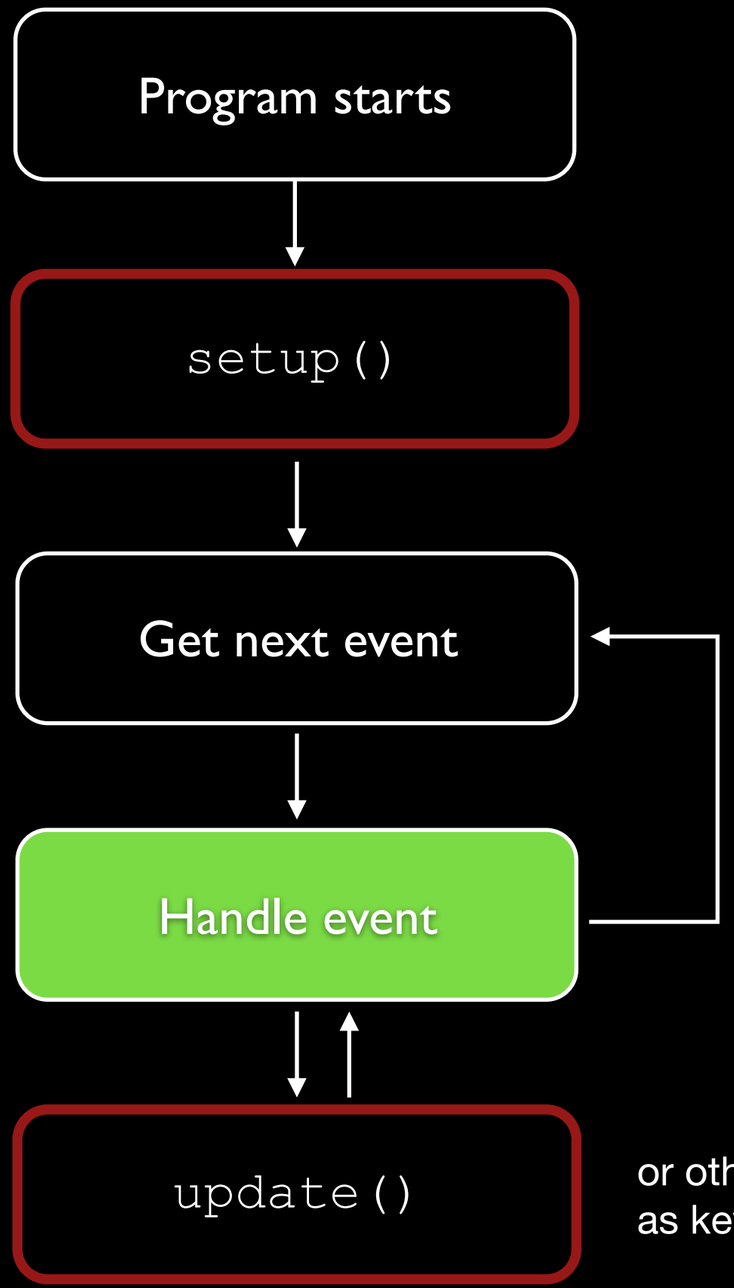
or other event handlers for things such as keyboard or mouse (pointer) events.



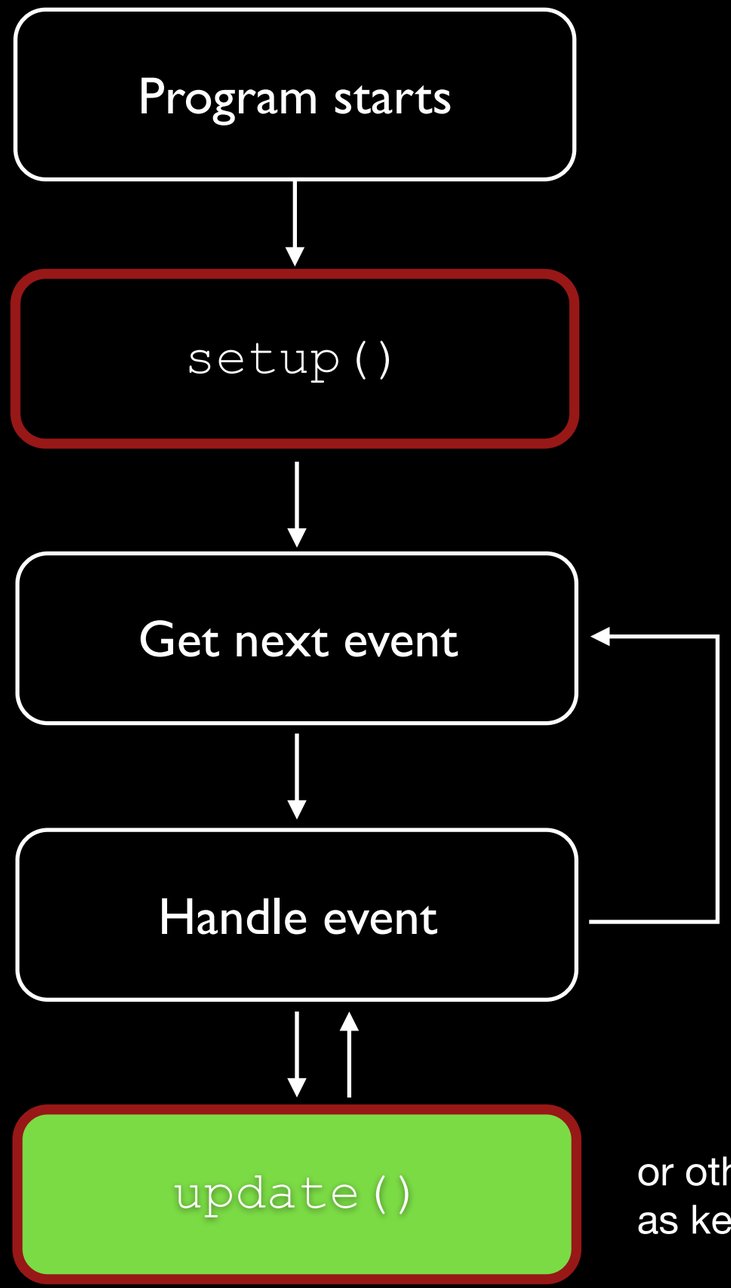
or other event handlers for things such as keyboard or mouse (pointer) events.



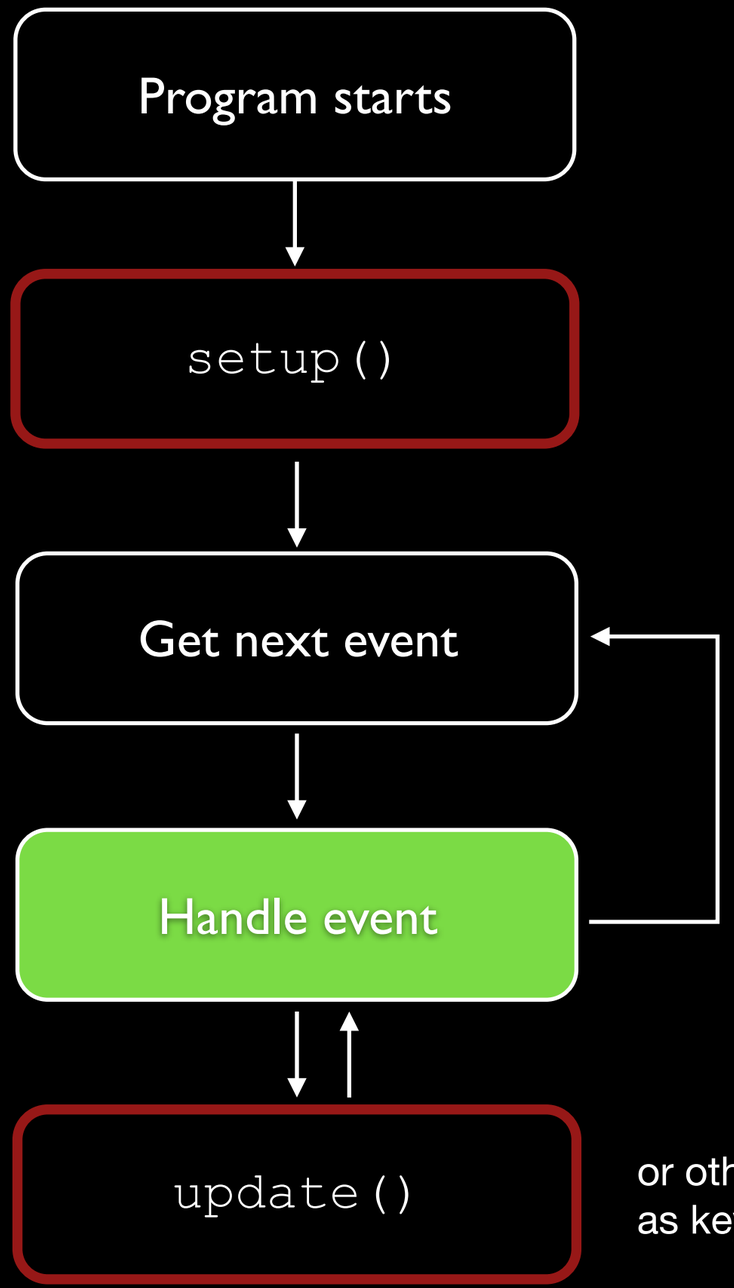
or other event handlers for things such as keyboard or mouse (pointer) events.



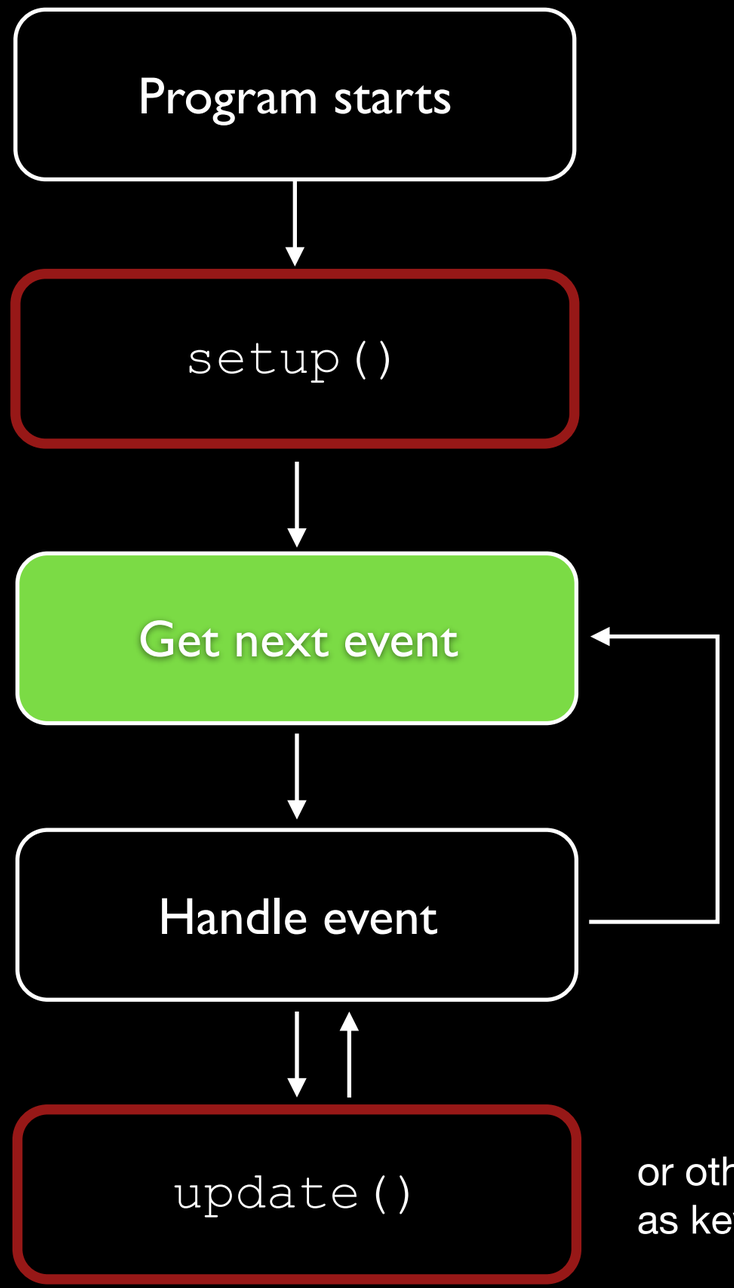
or other event handlers for things such as keyboard or mouse (pointer) events.



or other event handlers for things such as keyboard or mouse (pointer) events.



or other event handlers for things such as keyboard or mouse (pointer) events.



or other event handlers for things such as keyboard or mouse (pointer) events.

Tin Scene object

```
override func setup() {

  // put your instructions here

}
```

Called once when the program is started. Used to define initial properties values before the **update()** function is called.

Tin Scene object

```
override func update() {

  // put your instructions here

}
```

Called after **setup()** and the repeats at a set framerate until the program is stopped. The default framerate is 60 times per second. The **update()** function is called automatically and should never be called directly.

Code Blocks

```
{
  A block of code
  {
    A block inside a block of code
  }
}
```

These small curly bracket (or brace) characters { and } are ***very*** important. You wouldn't ignore a period (.) or comma (,) in your English writing class - would you? No, you would not. You must **ALWAYS** pay careful attention to any curly brackets in your code.

Please think about your curly brackets.

{

For example, a curly bracket is never a solo curly bracket in Swift.

}

Whenever there is an opening curly bracket {, there will **ALWAYS** be a matching (companion) closing curly bracket } that goes with it.

Consider the update function in this code example. Curly brackets define start and ending of the body of the function.

```
22
23  class Scene: TScene {
24
25      //
26      // The update function is called to draw the view automatically.
27      //
28      override func update() {
29          // background erases the view and sets the entire view to one flat
30          // color. If you want a different background color, change it here.
31          background(gray: 0.3)
32
33          // ************************************************
34          // Insert your drawing code here, below this comment
35
36          strokeDisable()
37
38          let x = 250.0
39          let y = 200.0
40
41          ///////eyeball
42          fillColor(red: 0.8, green: 0.6, blue: 0.6, alpha: 1.0)
43          ellipse(centerX: x + 30, centerY: y + 80, width: 80, height: 80)
44          fillColor(red: 0.0, green: 0.0, blue: 0.0, alpha: 1.0)
45          ellipse(centerX: x + 155, centerY: y + 110, width: 310, height: 220)
46          fillColor(red: 0.8, green: 0.8, blue: 0.8, alpha: 1.0)
47          ellipse(centerX: x + 150, centerY: y + 100, width: 300, height: 200)
48          fillColor(red: 0.0, green: 0.0, blue: 0.0, alpha: 1.0)
49          ellipse(centerX: x + 150, centerY: y + 100, width: 200, height: 200)
50
51          // iris
52          fillColor(red: 0.1, green: 0.3, blue: 0.7, alpha: 1.0)
53          ellipse(centerX: x + 150, centerY: y + 100, width: 180, height: 180)
54          fillColor(red: 0.3, green: 0.5, blue: 0.9, alpha: 1.0)
55          ellipse(centerX: x + 150, centerY: y + 85, width: 150, height: 150)
56          fillColor(red: 0.4, green: 0.6, blue: 1.0, alpha: 1.0)
57          ellipse(centerX: x + 150, centerY: y + 70, width: 120, height: 120)
58
59          // pupil
60          fillColor(red: 0.0, green: 0.0, blue: 0.0, alpha: 1.0)
61          ellipse(centerX: x + 150, centerY: y + 100, width: 100, height: 100)
62
63          //highlights
64          fillColor(red: 1.0, green: 1.0, blue: 1.0, alpha: 1.0)
65          ellipse(centerX: x + 75, centerY: y + 125, width: 50, height: 50)
66          ellipse(centerX: x + 192.5, centerY: y + 47.5, width: 25, height: 25)
67
68          // Your drawing code should be above this comment.
69          // ************************************************
70
71          view?.stopUpdates()
72      }
73
74  }
75
76
```

update Opening bracket.

body of update function

update Closing bracket.

Here is a different update function. It is common to have nested blocks of code. Blocks inside blocks that are inside other blocks!

Indent the code inside any set of curly brackets. The indentation makes it easier to understand which block the code is inside.

```
35
36      //
37      // The update function is called to draw the view automatically.
38      //
39      override func update() {
40          // background erases the view and sets the entire view to one flat
41          // color. If you want a different background color, change it here.
42          background(gray: 0.5)
43
44
45          // Location of the mouse
46          let mouseX = tin.mouseX
47          let mouseY = tin.mouseY
48
49          // Size of rectangle for drawing
50          let w = tin.width / 2
51          let h = tin.height / 2
52
53          strokeDisable()
54          if mouseX < tin.midX && mouseY < tin.midY {
55              // Mouse is in the bottom, left quadrant.
56              fillColor(red: 1, green: 0, blue: 0, alpha: 1)
57              rect(x: 0, y: 0, width: w, height: h)
58          }
59          else if mouseX > tin.midX && mouseY < tin.midY {
60              // Mouse is in the bottom, right quadrant
61              fillColor(red: 0, green: 1, blue: 0, alpha: 1)
62              rect(x: tin.midX, y: 0, width: w, height: h)
63          }
64          else if mouseX > tin.midX && mouseY > tin.midY {
65              // Mouse is in the top, right quadrant
66              fillColor(red: 0, green: 0, blue: 1, alpha: 1)
67              rect(x: tin.midX, y: tin.midY, width: w, height: h)
68          }
69          else {
70              // Mouse is in the top, left quadrant
71              // We know that it is in top, left only because if it is not
72              // in any of the others, it must be in the top, left.
73              fillColor(red: 1, green: 1, blue: 0, alpha: 1)
74              rect(x: 0, y: tin.midY, width: w, height: h)
75          }
76
77      }
78
79  }
80
```

A block inside update.
body of update function

A block inside update.

A block inside update.

A block inside update.

Please think about your curly brackets.

Pay close attention to your curly brackets when you edit code. They are relatively small, and its easy to miss them. Just like its easy to miss a period at the end of a sentence.

Tin mouse pointer location

The mouse pointer location can be obtained using two properties of the tin object:

```
tin.mouseX
tin.mouseY
```

Draw a circle at the mouse pointer location:

```
ellipse(centerX: tin.mouseX, centerY: tin.mouseY, width:30.0, height: 30.0)
```