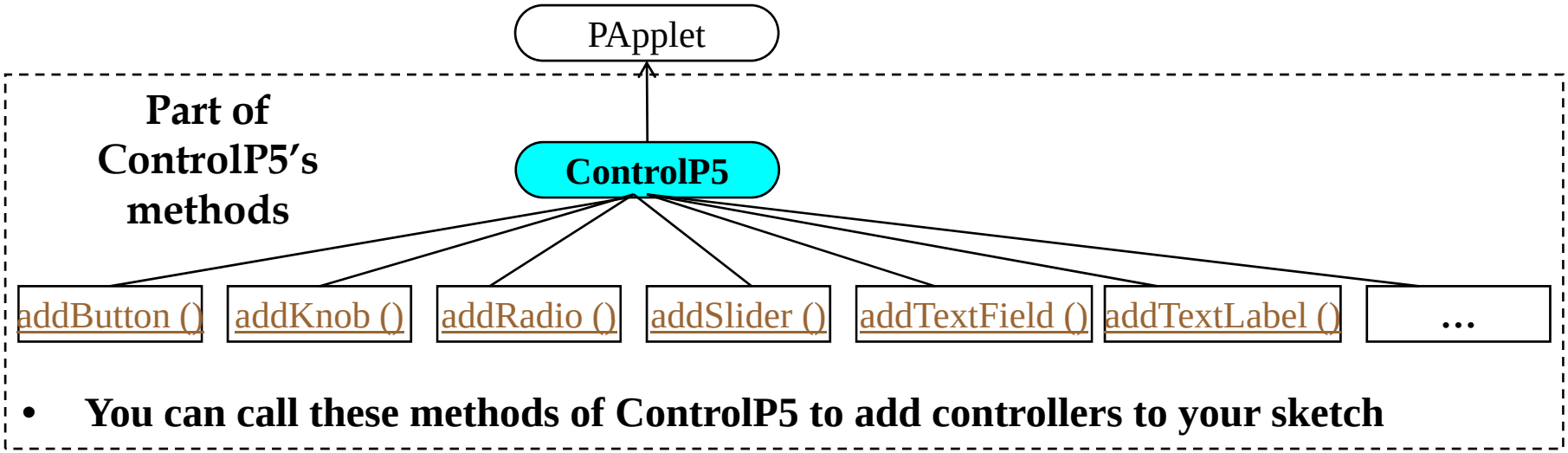
IAT 265

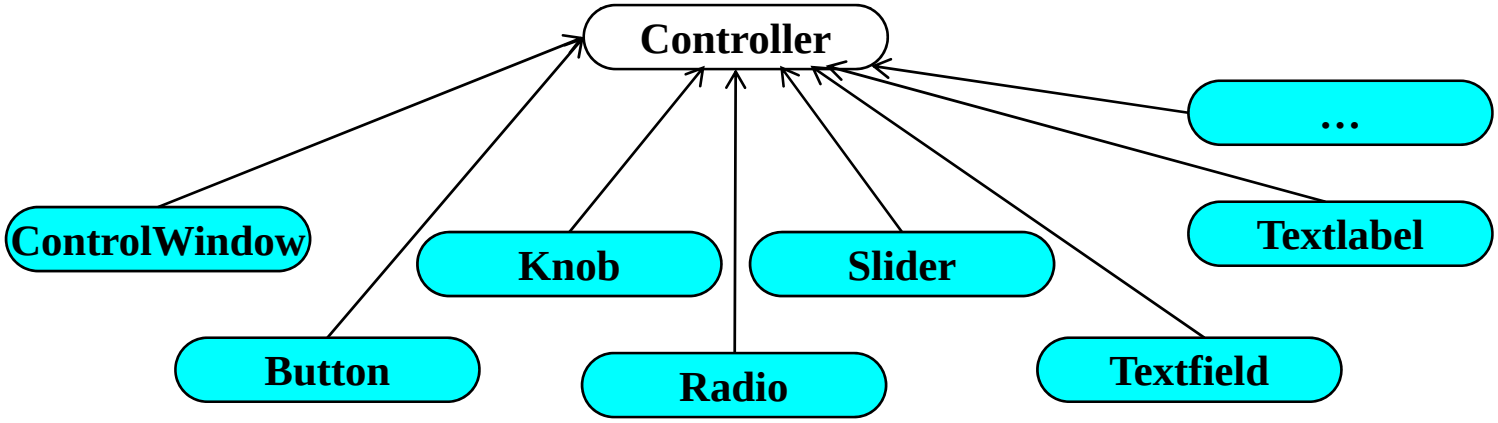
Recursion and Graphics

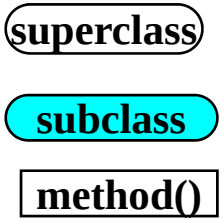
Today's Topics

- Review some key concepts for GUIs
- Handle events from multiple controllers
- Recursion
- Recursion and Graphics



- **You can call these methods of ControlP5 to add controllers to your sketch**





- **Controller is the superclass of all available controllers …**

- **You can call each controller's methods (including inherited) to set or get its properties such as label, value, color etc., or do other manipulations**

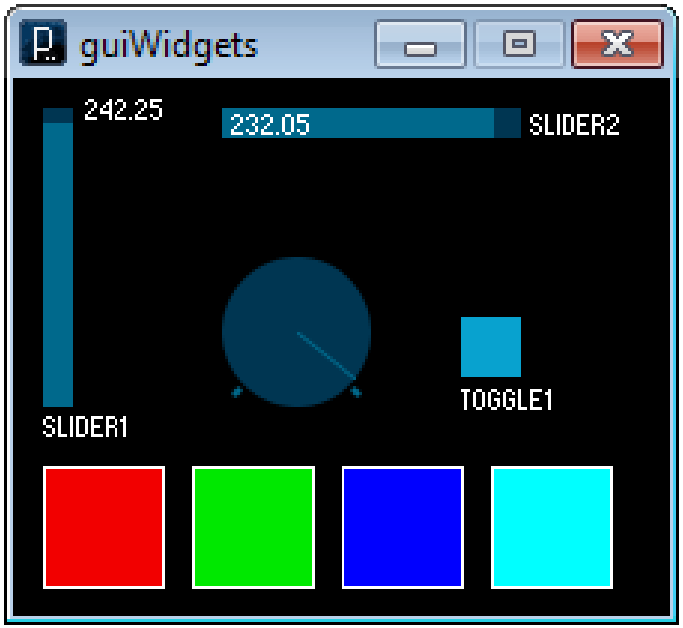
Review: Event-Driven Programming Model

- Program waits for *events* to occur and then responds, which is divided down to three sections:
  - ***Event firing*** – objects/interactions generate events
  - ***Event detection*** – listeners check for events
    - Normally taken care of by program frameworks
  - ***Event handling*** – functions respond to events
    - Programmer need to define these functions (aka event-handlers), typically they are ***callbacks***

Handle Events from Multiple Controllers

- Modify the colors in the color array with different controllers, and display them in the rectangles

```
This will require you to:

1) add those controllers
   to the window

2) handle events fired
   from these controllers

3) respond by changing the
   color squares as per
   which controller is
   changed
```



Implementation

1. Create ***ControlP5*** object, color array, & squares

```
import controlP5.*;          //import the whole library
ControlP5 controlP5;         //declare variable of ControlP5
//array of colors that can be changed by controllers
color [] colors = new color[4];
void setup() {
  size(220,180);
  smooth();
  controlP5 = new ControlP5(this);
}

void draw() {
  background(0);
 //draw 4 squares with colors from the colors array
  for(int i=0;i<colors.length;i++) {
    stroke(255);
    fill(colors[i]);
    rect(10+(i*50),130,40,40);  // draw a rectangle
  }
}
```

Implementation

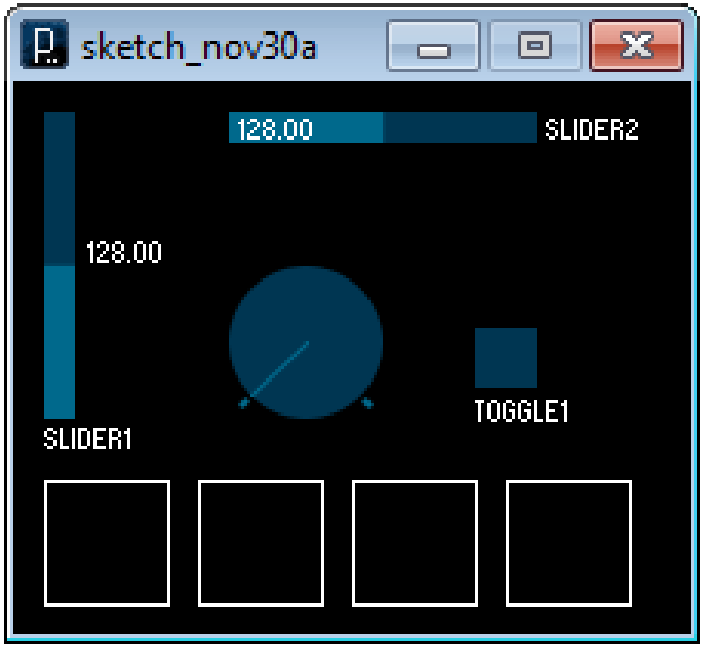
2. Add the controllers

Can you tell which is horizontal or vertical?

```
void setup() {
  …    //whatever was done so far
 //Add Sliders. Parameters: name, minimum, maximum,
 //default value (float), x, y, width, height
  controlP5.addSlider("slider1",0,255,128,10,10,10,100);
  controlP5.addSlider("slider2",0,255,128,70,10,100,10);

  //Add a Knob. Parameters: name, minimum, maximum,
  //default value (float), x, y, diameter
  controlP5.addKnob("knob1",0,360,0,70,60,50);

  //Add a toggle which have two states: true or false
  //parameters: name, default value (boolean), x, y,
  //width, height
  controlP5.addToggle("toggle1",false,150,80,20,20);
}
```

Run the sketch and this is what you got so far...

- The controllers are all in place, and visually functional, however it brings no change to squares' color. Why?



Handle Events from Multiple Controllers

- That's because we haven't created any code to handle events fired from those controllers when you manipulate them

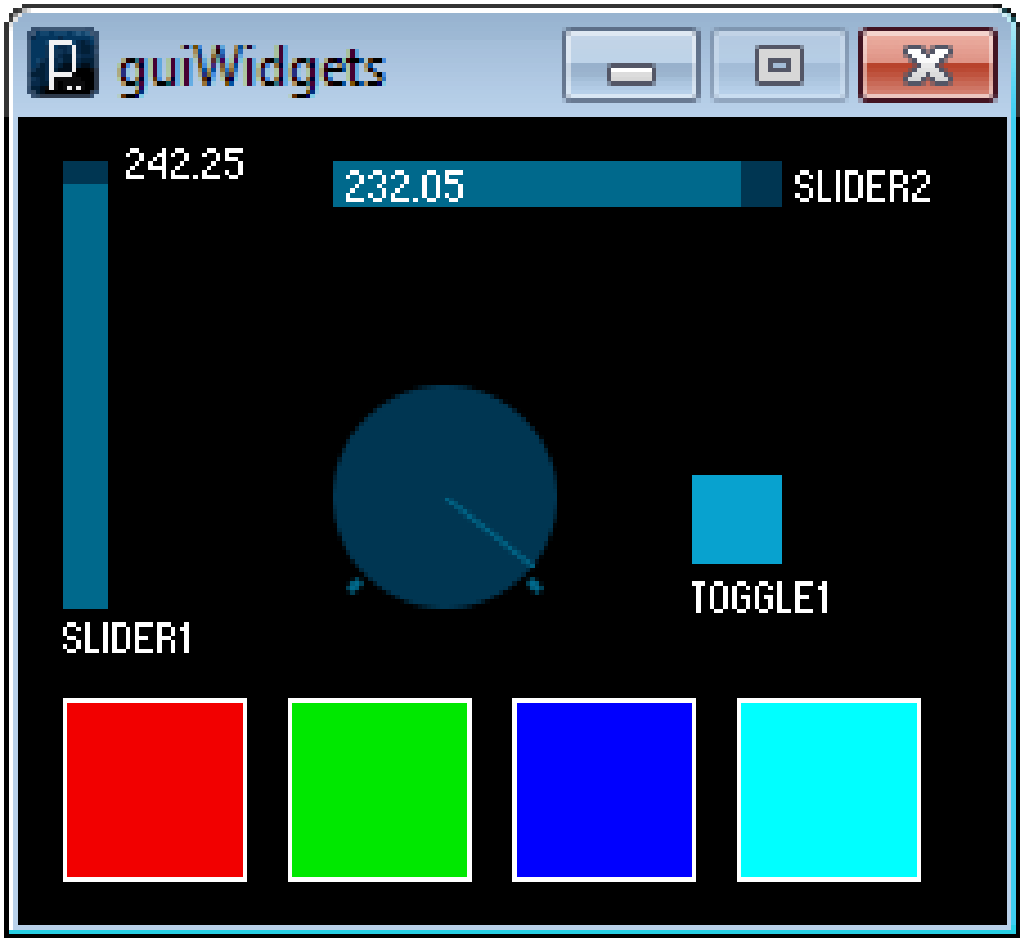
- This is what we need to do:
  - add the event handler: ***controlEvent(theEvent)***
  - Inside it, use multiple ***if-statements***, one for each controller, to detect which has been changed
  - Then respond by changing the corresponding square's color accordingly

Implementation

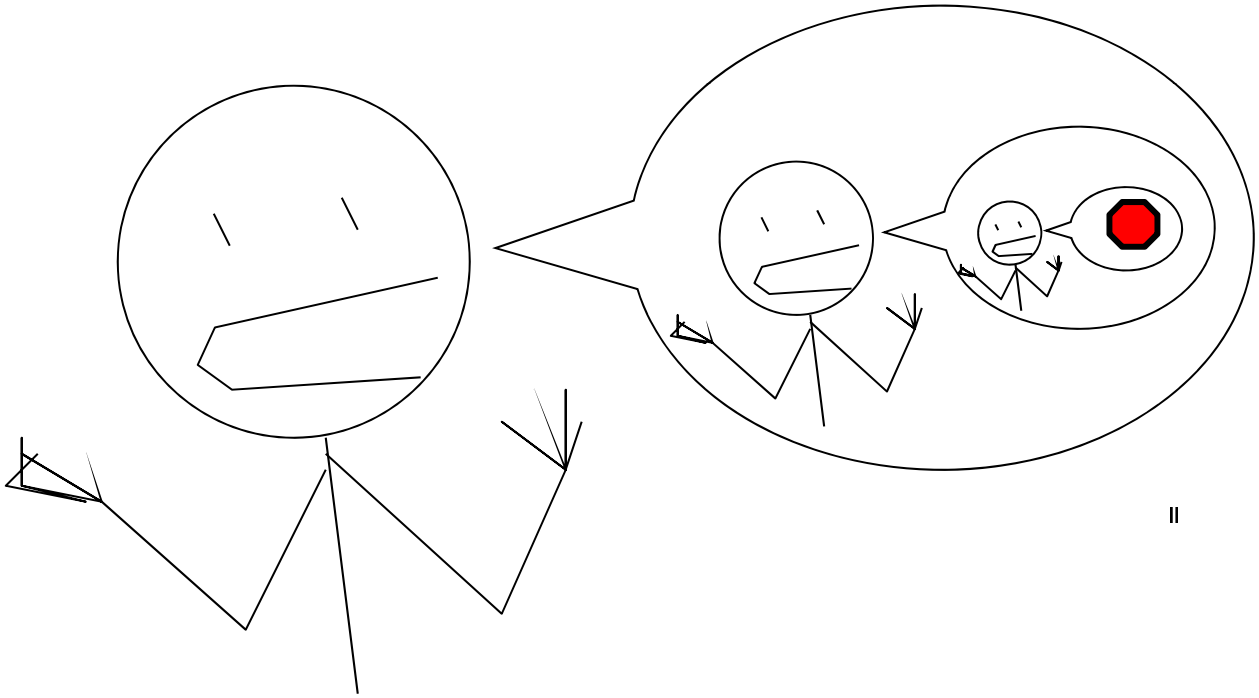
3. Add the event handler: ***controlEvent(theEvent)***

```
 …     //whatever is done so far
void controlEvent(ControlEvent theEvent) {
   if(theEvent.controller().name()=="slider1")
      colors[0] = color(theEvent.controller().value(),0,0);

   if(theEvent.controller().name()=="slider2")
      colors[1] = color(0,theEvent.controller().value(),0);

   if(theEvent.controller().name()=="knob1")
      colors[2] = color(0,0,theEvent.controller().value());

   if(theEvent.controller().name()=="toggle1") {
      if(theEvent.controller().value()==1)
         colors[3] = color(0,255,255); //set color to cyan
      else
         colors[3] = color(0,0,0);     //otherwise black
   }
}
```

This is what you end up with...



Recursion



Recursion

- Recursion basically means calling a method from inside itself

```
int factorial(int n)
{
  if( n > 1 )
  {
    return(  n* factorial( n-1 )  );
  }
  else
    return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);
      ↓
int factorial(int n)   (n=3)
{
        if(n > 1)
        {
            return( n* factorial( n-1 ) );
        }
        else
            return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=2)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=2)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=1)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) )
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=2)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=1)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) )
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=2)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

1

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=2)
{
 if(n > 1)
 {
   return( n* 1 );
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

```
int factorial(int n)  (n=2)
{
 if(n > 1)
 {
   return( 2* 1 );
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);

int factorial(int n)   (n=3)                    2
{
 if(n > 1)
 {
   return( n* factorial( n-1 ) );
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);
        ↓
int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( n* 2 );
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

```
factorial(3);
        ↓
int factorial(int n)   (n=3)
{
 if(n > 1)
 {
   return( 3* 2 );
 }
 else
   return( 1 );
}
```

Calling Itself

- Let's step through what happens

factorial(3);

↓

6

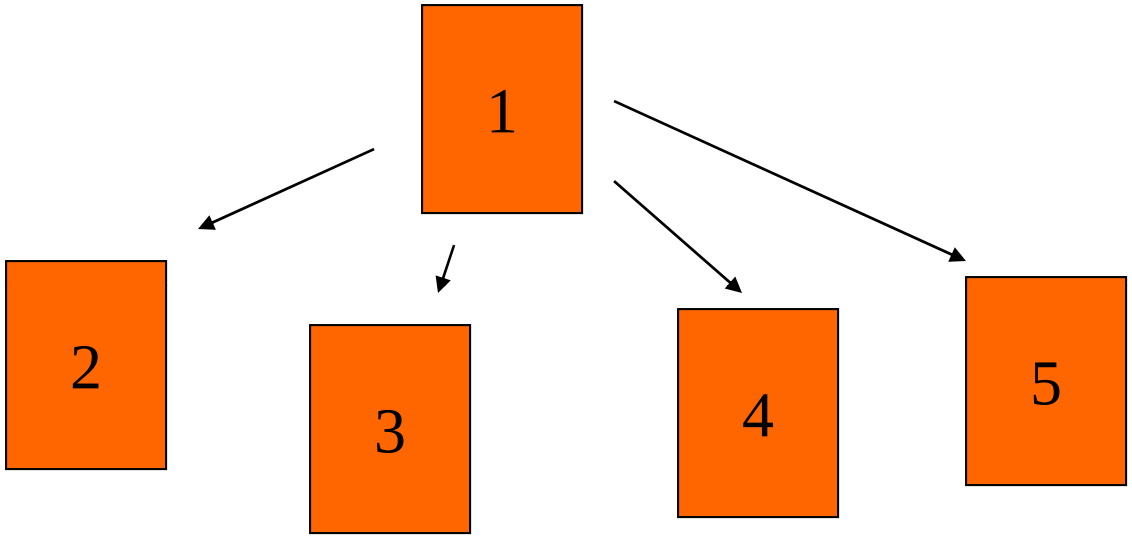
Base Case

- Must have at least one **Base Case**
  - A case or condition that returns without further recursion
    - Stops the recursive chain
  - Eg factorial(int n)
    - Returned 1 when n = 1 //base case
    - In every other call, n decreases by 1

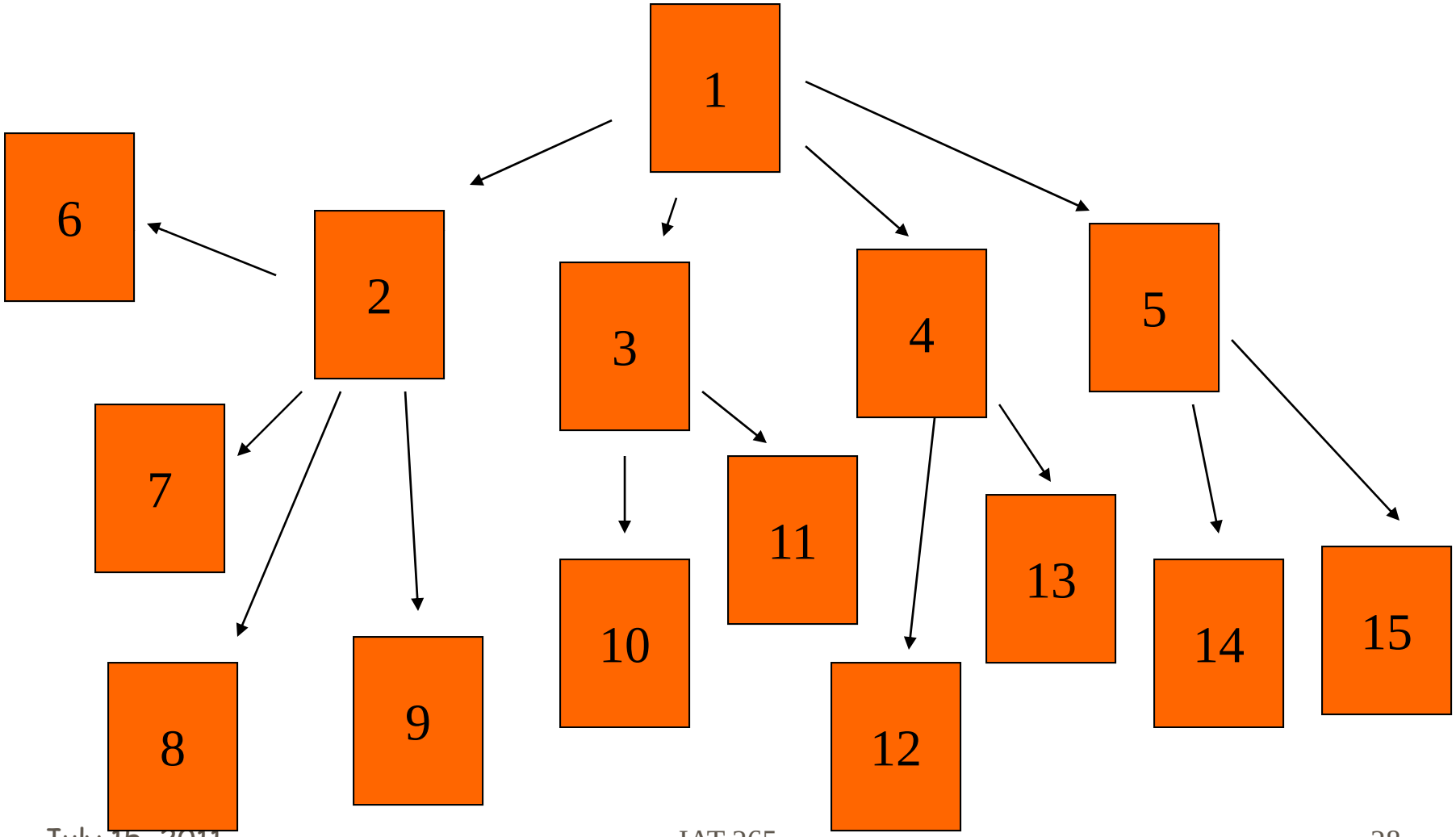
An application of recursion: Web Crawling

- HTML reader called **parsePage()**
  - Reads HTML
  - Finds links
  - For each Link it should
    - Call **parsePage()**

Web Crawling



Web Crawling



Web Crawling

- What is the base case?
  - Count the number of recursive calls so far
  - Place a limit on **depth**
    - E.g. explore no further after depth 4

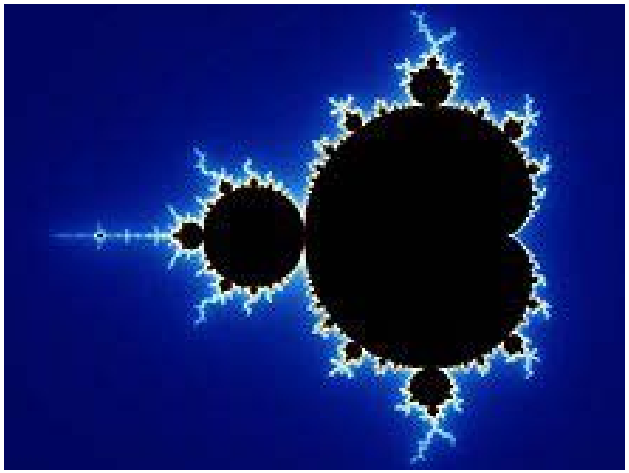
Recursion and Graphics

- So far we have dealt with shapes that can be described by idealized geometrical forms – rectangle, ellipse, arc, ...
  - Parts have no similarity with the whole shape

- However there is another type of shape - *fractal* shapes
  - Mandelbrot coined the term to describe self-similar shapes

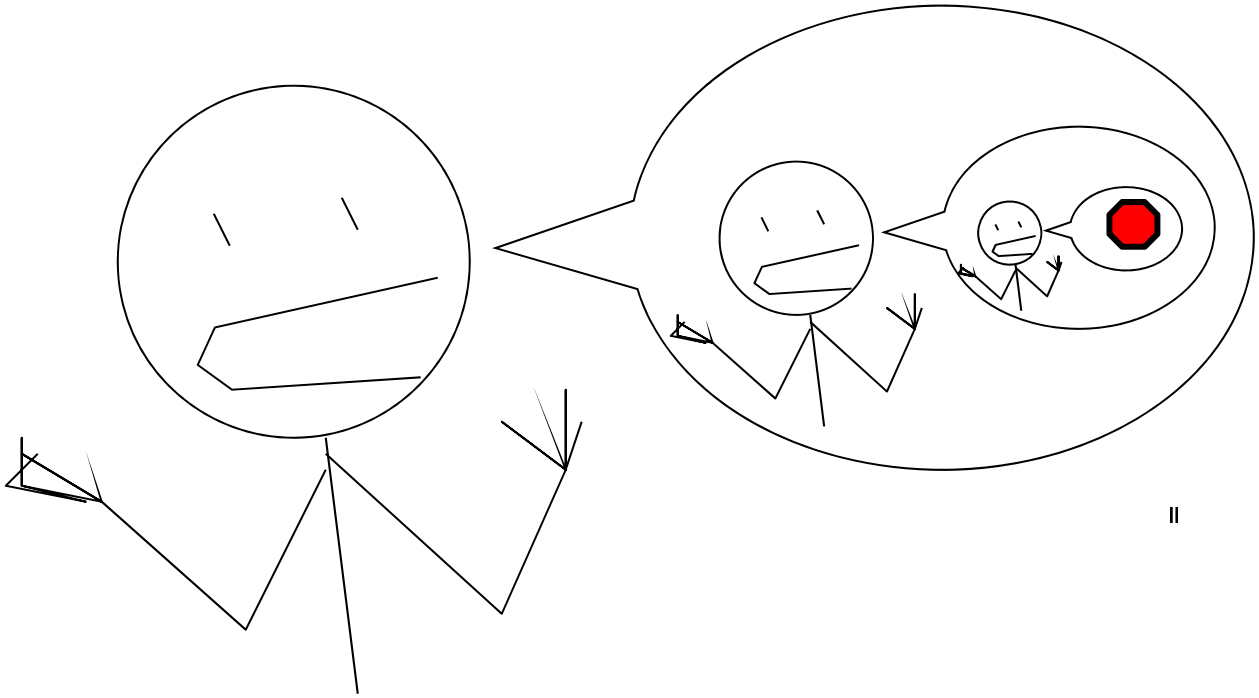
Fractal shapes

- Mandelbrot: “A rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole“



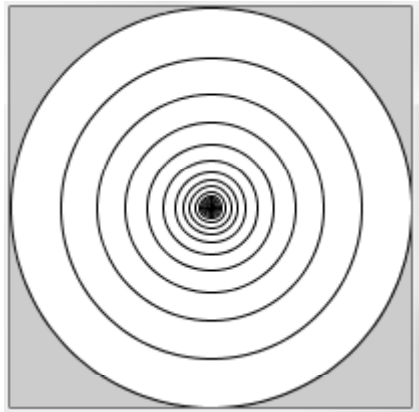
- Examples are snowflakes, trees, coastlines, mountains, ...

Recursion is a good instrument for generating fractals



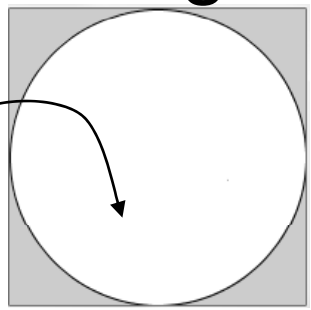
A simple example

```
void drawCircle (float x, float y, float d) { //d - diameter
    ellipse(x, y, d, d);

    if (d > 2) {
        d *= 0.75; //shrink d by 25% each recursion
        drawCircle (x, y, d);
    }
}

void setup(){
  size(200, 200);
  drawCircle (width/2, height/2, width); //drawCircle(100, 100, 200);
}
```



Let's step through what happens

```
drawCircle(100, 100, 200);
```

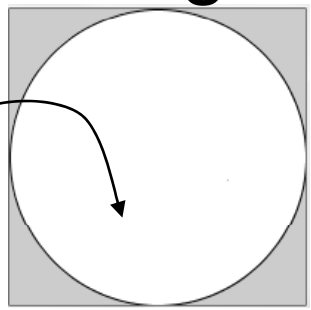


```
void drawCircle (float x, float
y, float d) (d=200)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```

```
void drawCircle (float x, float y,
float d) (d=100)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```

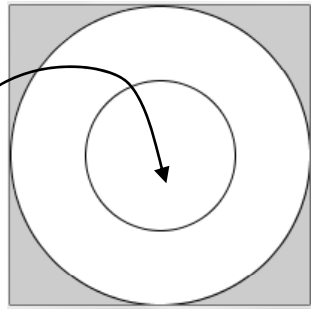
Let's step through what happens

```
drawCircle(100, 100, 200);
```



```
void drawCircle (float x, float
y, float d) (d=200)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```

```
void drawCircle (float x, float
y, float d) (d=100)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```



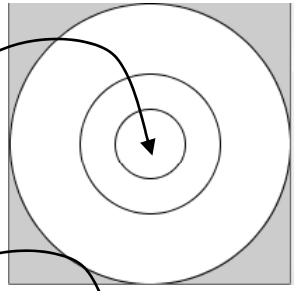
```
void drawCircle (float x, float
y, float d) (d=50)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```

Let's step through what happens

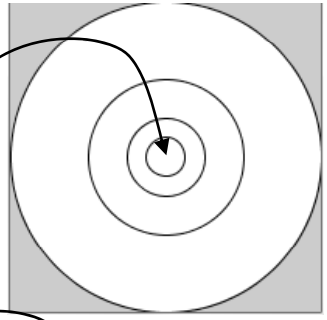
```
drawCircle(100, 100, 200);
```

↓

```
void drawCircle (float x, float
y, float d) (d=50)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```



```
void drawCircle (float x, float
y, float d) (d=25)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```

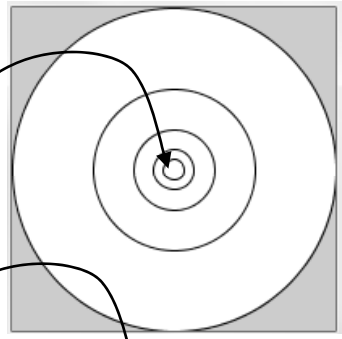


```
void drawCircle (float x, float
y, float d) (d=12.5)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```

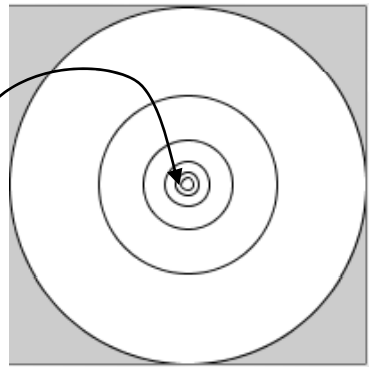
Let's step through what happens

drawCircle(100, 100, 200);

```
void drawCircle (float x, float
y, float d) (d=12.5)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```



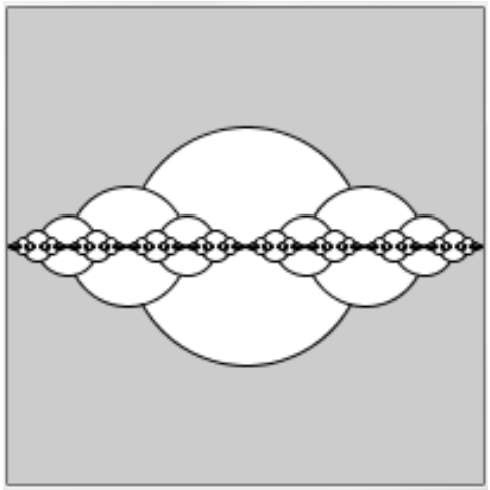
```
void drawCircle (float x, float
y, float d) (d=6.25)
{
   ellipse(x, y, d, d);

   if (d > 10) {
      d *= 0. 5;
      drawCircle(x, y, d);
   }
}
```



d = 6.25 < 10 → base case hit, so stop recursive call, start to return

Branching Effect

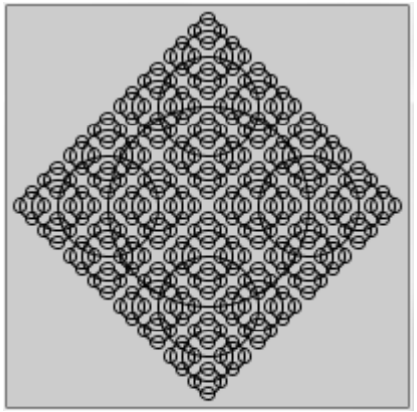
```
void drawCircle (float x, float y, float diameter) {
  ellipse(x, y, diameter, diameter);
  if (diameter > 2) {
    diameter *= 0.5;
    drawCircle(x+diameter, y, diameter);
    drawCircle(x-diameter, y, diameter);
  }
}
```



```
void setup(){
  size(200, 200);
  drawCircle (width/2, height/2, width/2);    //drawCircle(100, 100, 100);
}
```

More sophisticated Fractal

```
void drawCircle (float x, float y, float diameter) {
  ellipse(x, y, diameter, diameter);
  if (diameter > 8) {
    drawCircle(x+diameter/2, y, diameter/2);
    drawCircle(x-diameter/2, y, diameter/2);
    drawCircle(x, y+diameter/2,  diameter/2);
    drawCircle(x, y-diameter/2, diameter/2);
  }
}
void setup(){
  size(200, 200);
  noFill();
  drawCircle(width/2, height/2, 100);
}
```



Summary

- Handle events from multiple controllers
  - Use multiple *if-statements*, one for each controller, to check which one is changed, and respond accordingly
- Recursion
  - A method calls itself (a recursive chain that stops at a base case)
- Recursion and Graphics
  - Good for generating fractals – self-similar shapes