# Chapter 5
# Views: Redrawing UIScrollView

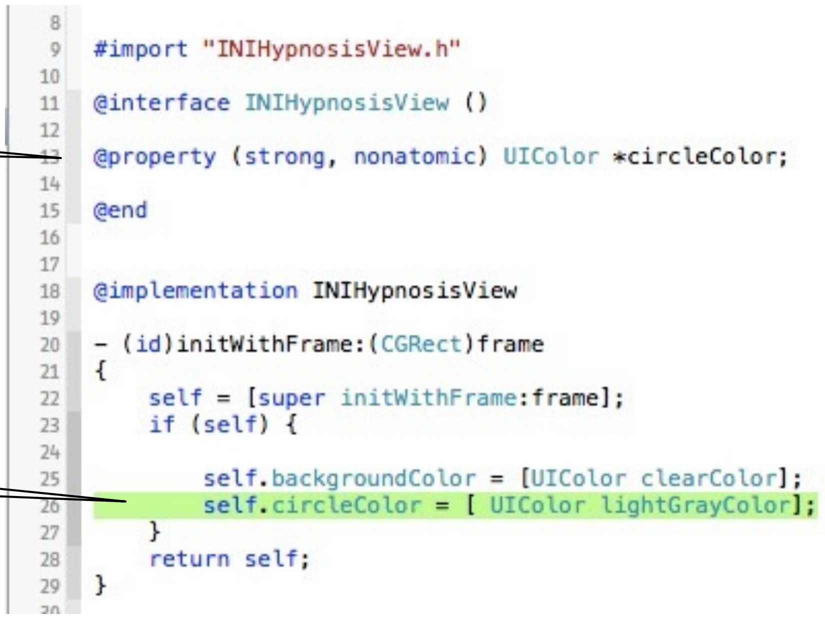
- Views response to events
- Using UIScrollViews

## Custom Circle Color

- Recall: We can define attributes as properties in an extension of the interface section housed in the implementation file.
- Such properties are private attributes.

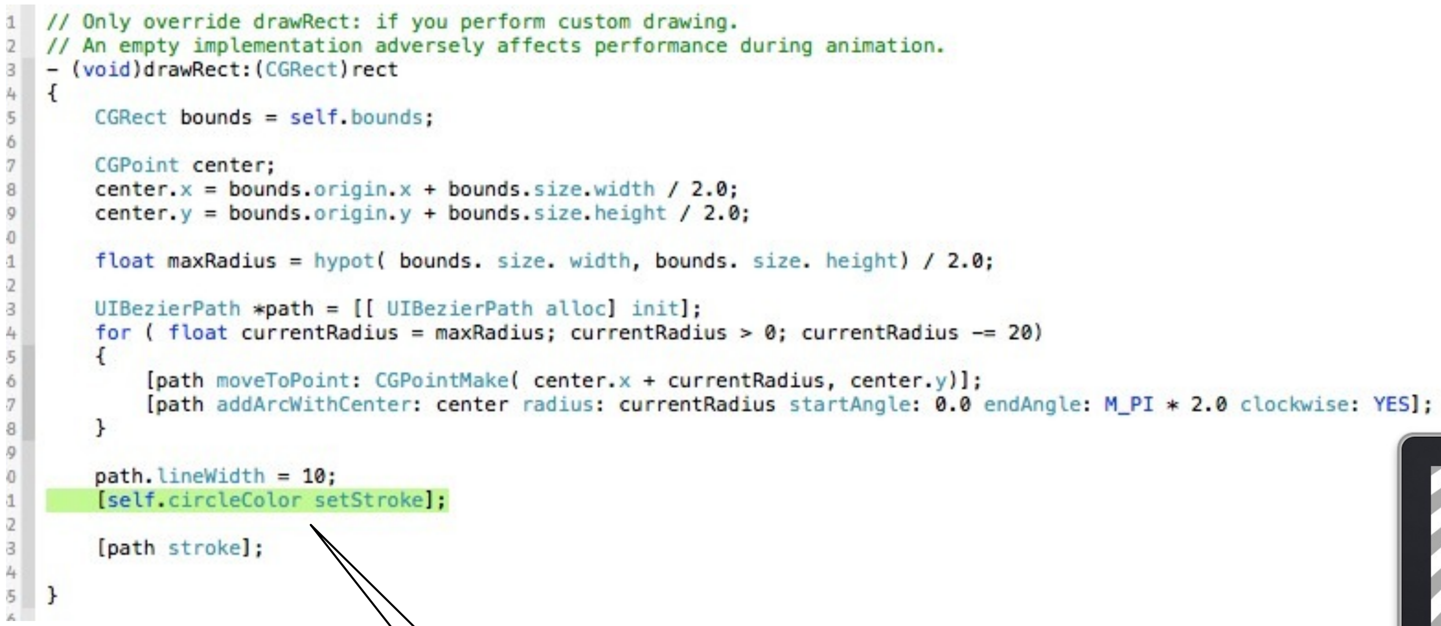
Defining circle color property

Setting circle color property to light gray



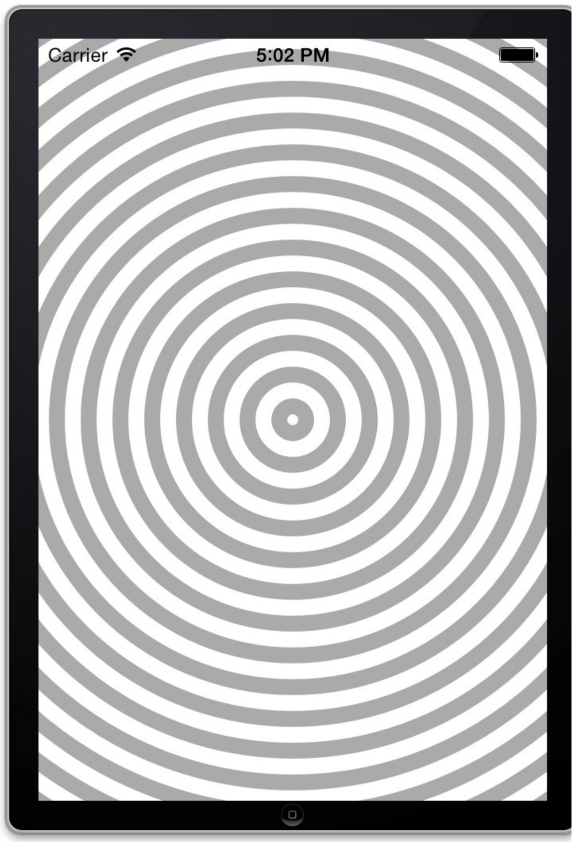
```
#import "INIHypnosisView.h"

@interface INIHypnosisView ()

@property (strong, nonatomic) UIColor *circleColor;

@end


@implementation INIHypnosisView

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {

        self.backgroundColor = [UIColor clearColor];
        self.circleColor = [ UIColor lightGrayColor];
    }
    return self;
}
```

## Update drawRect:



```
// Only override drawRect: if you perform custom drawing.
// An empty implementation adversely affects performance during animation.
- (void)drawRect:(CGRect)rect
{
    CGRect bounds = self.bounds;

    CGPoint center;
    center.x = bounds.origin.x + bounds.size.width / 2.0;
    center.y = bounds.origin.y + bounds.size.height / 2.0;

    float maxRadius = hypot( bounds. size. width, bounds. size. height) / 2.0;

    UIBezierPath *path = [[ UIBezierPath alloc] init];
    for ( float currentRadius = maxRadius; currentRadius > 0; currentRadius -= 20)
    {
        [path moveToPoint: CGPointMake( center.x + currentRadius, center.y)];
        [path addArcWithCenter: center radius: currentRadius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES];
    }

    path.lineWidth = 10;
    [self.circleColor setStroke];

    [path stroke];

}
```

Set stroke color to circleColor

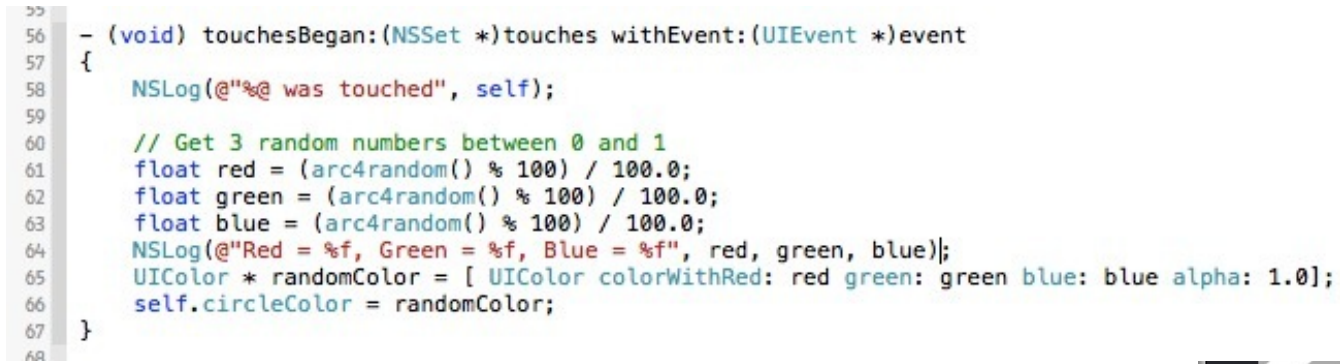


## INIHypnosisView Respond to Touch

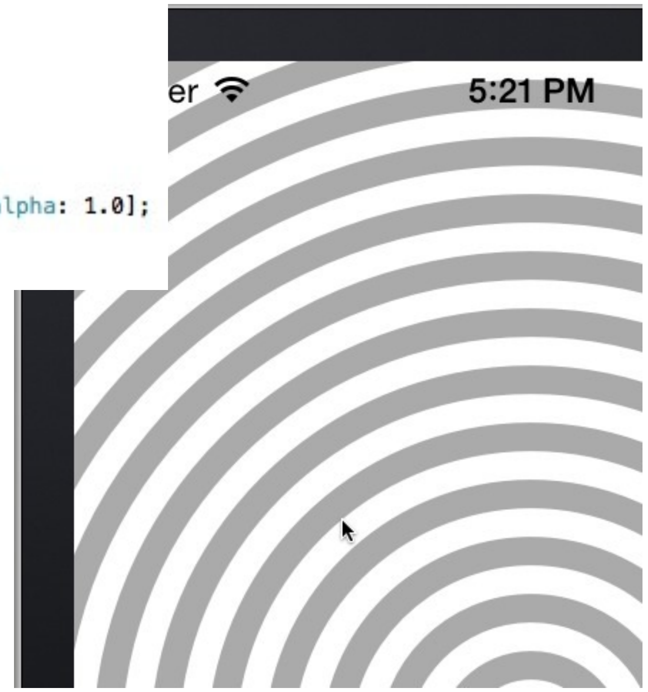
- When the user touches a view, the view is sent the message touchesBegan: withEvent:.
- The touchesBegan: withEvent: method is a touch event handler.
- We will need to override touchesBegan: withEvent: to change the circleColor property of the view to a random color.
- We will use arc4random() as a random number generator.
- To get an integer value from arc4random() that goes from 0 to x-1, you would do this:

```
int value = arc4random() % x;
```

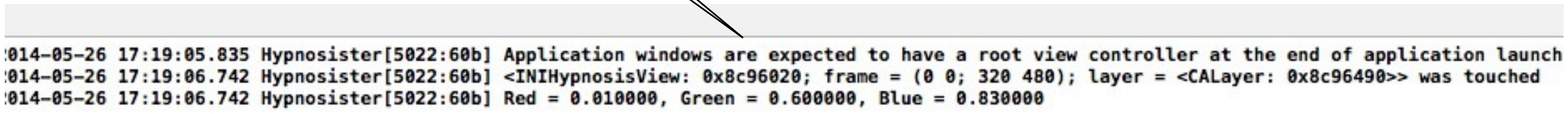
# Touching the screen



```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    NSLog(@"%@ was touched", self);

    // Get 3 random numbers between 0 and 1
    float red = (arc4random() % 100) / 100.0;
    float green = (arc4random() % 100) / 100.0;
    float blue = (arc4random() % 100) / 100.0;
    NSLog(@"Red = %f, Green = %f, Blue = %f", red, green, blue);
    UIColor * randomColor = [ UIColor colorWithRed: red green: green blue: blue alpha: 1.0];
    self.circleColor = randomColor;
}
```



It responds to the
touch but color does not change



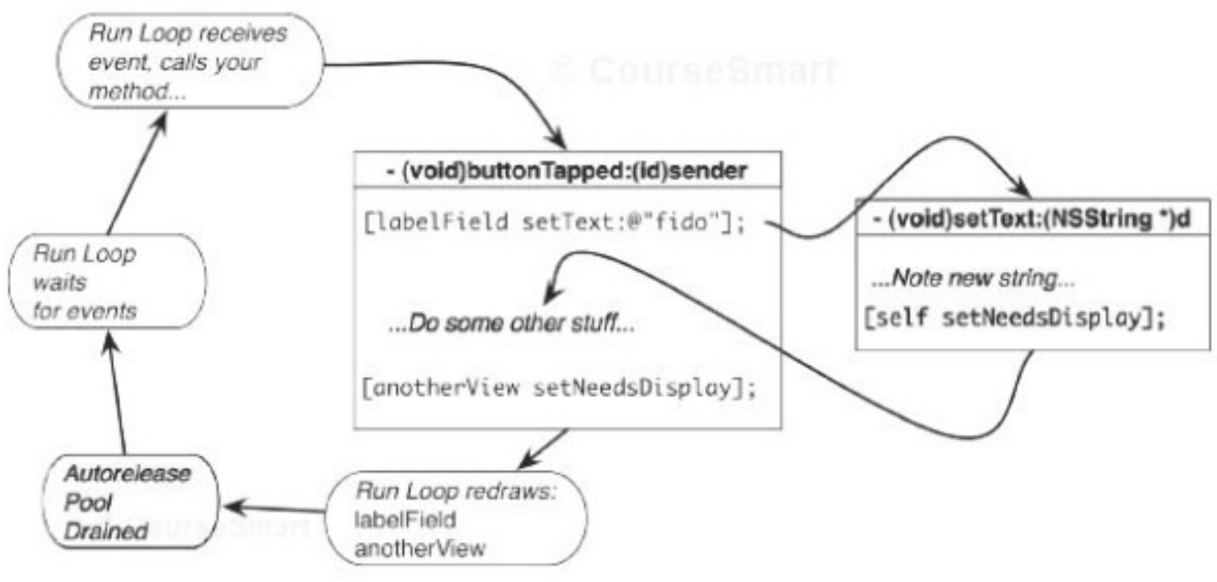
```
014-05-26 17:19:05.835 Hypnosister[5022:60b] Application windows are expected to have a root view controller at the end of application launch
014-05-26 17:19:06.742 Hypnosister[5022:60b] <INIHypnosisView: 0x8c96020; frame = (0 0; 320 480); layer = <CALayer: 0x8c96490>> was touched
014-05-26 17:19:06.742 Hypnosister[5022:60b] Red = 0.010000, Green = 0.600000, Blue = 0.830000
```

## The Run Loop and event handling

- When an iOS application is launched, it starts a run loop.
- The run loop’s job is to listen for events, such as a touch.
- When an event occurs, the run loop then finds the appropriate handler methods for the event.
- Those handler methods call other methods, which call more methods, and so on.
- Once all of the methods have completed, control returns to the run loop.
- When the run loop regains control, it checks a list of “ dirty views” – views that need to be re- rendered based on what happened in the most recent round of event handling.
- The run loop then sends the drawRect: message to the views in this list before all of the views in the hierarchy are composited together again.

## Using Run Loop to Redraw Views

- There are two optimizations:
  - only re- rendering views that need it
  - Only sending drawRect: once per event
- These two optimizations keep iOS interfaces responsive.
- If iOS applications had to redraw every view every time an event was processed, there would be a lot of time wasted doing unnecessary work.

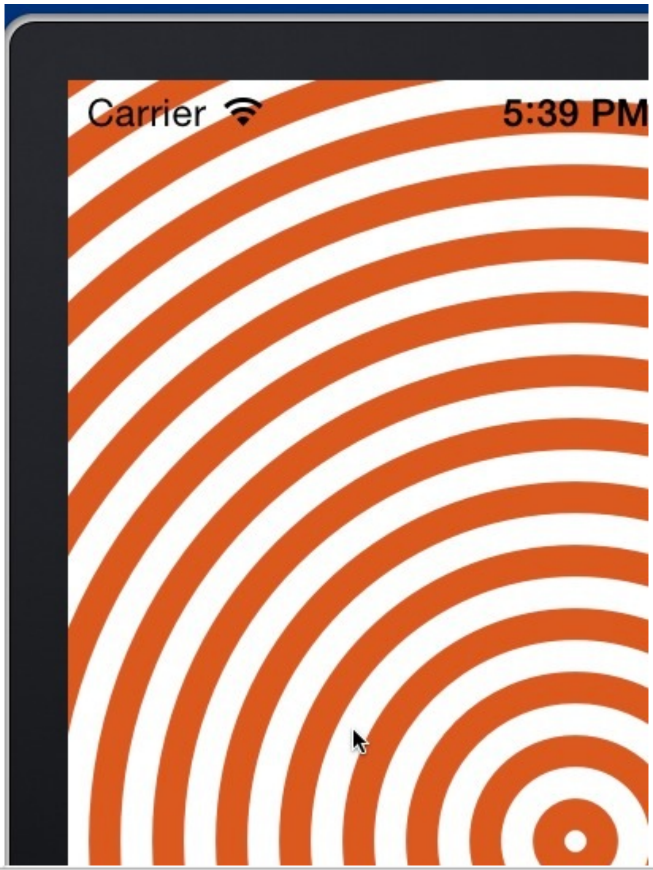


Run Loop receives event, calls your method...
- (void)buttonTapped:(id)sender
[labelField setText:@"fido"];
...Do some other stuff...
[anotherView setNeedsDisplay];
- (void)setText:(NSString *)d
...Note new string...
[self setNeedsDisplay];
Run Loop redraws: labelField anotherView
Autorelease Pool Drained
Run Loop waits for events
© CourseSmart

## Getting on the Dirty List

- To get a view on the list of dirty views, you must send it the message setNeedsDisplay.
- The subclasses of UIView that are part of the iOS SDK send themselves setNeedsDisplay whenever their content changes.
- For example, an instance of UILabel will send itself setNeedsDisplay when it is sent setText:, since changing the text of a label requires the label to re-render its layer.
- In custom UIView subclasses, like INIHypnosisView, you must send this message yourself.
- In INIHypnosisView.m, implement a custom accessor for the circleColor property to send setNeedsDisplay to the view whenever this property is changed.

```
- (void) setCircleColor:( UIColor *) circleColor
{
    // This method will be called every time we set circleColor to new color as in "self.circleColor = randomColor;"
    _circleColor = circleColor;
    [ self setNeedsDisplay];
}
```



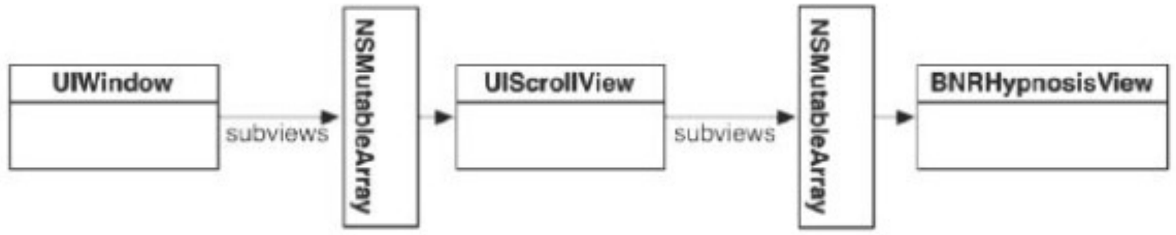
```
2014-05-26 17:39:39.104 Hypnosister[5224:60b] Application windows are expected to have a root view controller at the end of application launc
2014-05-26 17:39:40.293 Hypnosister[5224:60b] <INIHypnosisView: 0x8f09860; frame = (0 0; 320 480); layer = <CALayer: 0x8f098d0>> was touched
2014-05-26 17:39:40.293 Hypnosister[5224:60b] Red = 0.000000, Green = 0.030000, Blue = 0.070000
2014-05-26 17:39:42.924 Hypnosister[5224:60b] <INIHypnosisView: 0x8f09860; frame = (0 0; 320 480); layer = <CALayer: 0x8f098d0>> was touched
2014-05-26 17:39:42.925 Hypnosister[5224:60b] Red = 0.850000, Green = 0.350000, Blue = 0.170000
```

## Class Extensions

- Recall that the circleColor property that we declared in a class extension for INIHypnosisView.
- The difference between a property declared in a class extension and one declared in a header file is the visibility.
- A class's header file is visible to other classes and is used to advertise to other classes how they can interact with the class or its instances.
- Properties and methods that are used internally by the class do not need to be advertised and thus belong in a class extension.
- The circleColor property is only used by the INIHypnosisView class and no other class needs to know about this property. Thus, it belongs in the class extension.
- The same visibility rules hold for subclasses.
- If you were to subclass INIHypnosisView, the subclass and its instances would not know about circleColor.
- If you need limited visibility for certain properties and methods, you can create a class extension in an external file and import it into the implementation files of classes on a need- to- know basis.

## Using UIScrollView

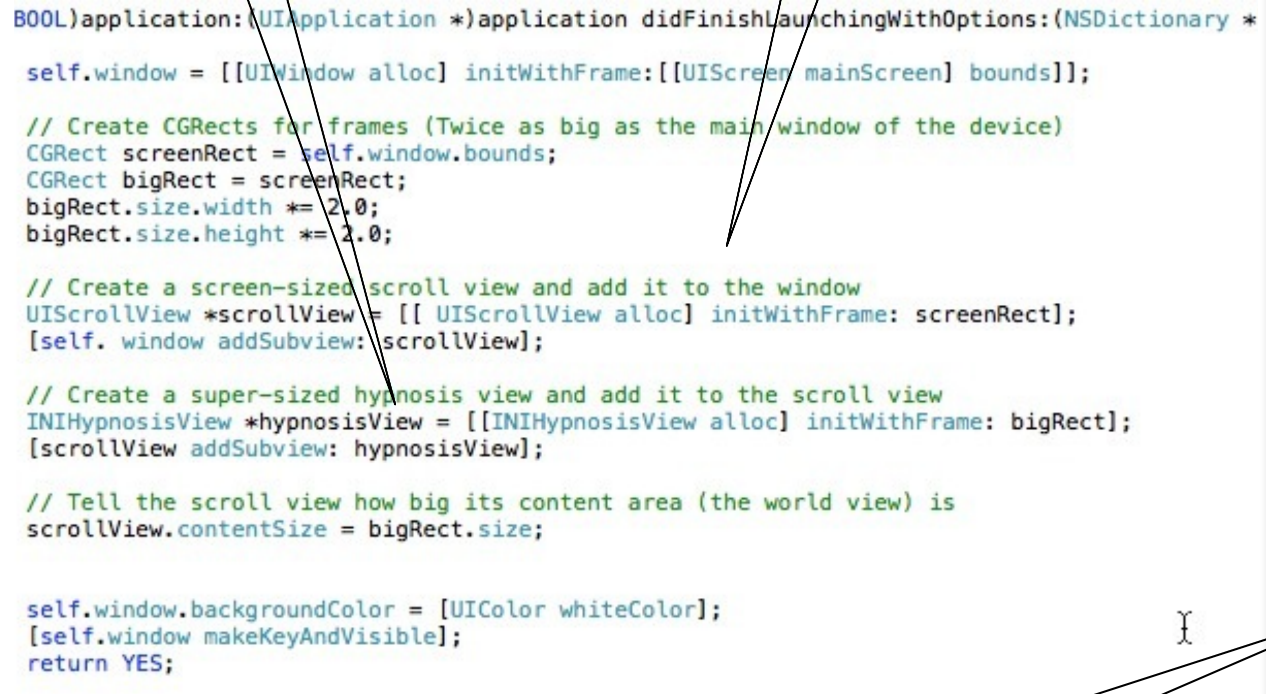
- A scroll view are used to view views that are larger than the screen.
- A scroll view is a viewing port that the user can move around a virtual world.
- There are two sizes that we associate with a scroll view:
    - The size of the viewing port (Typically this is the screen size).
    - The size of the world we are viewing and this is called the content size
- The views hierarchy is : UIwindow→UIScrollView→Your World View



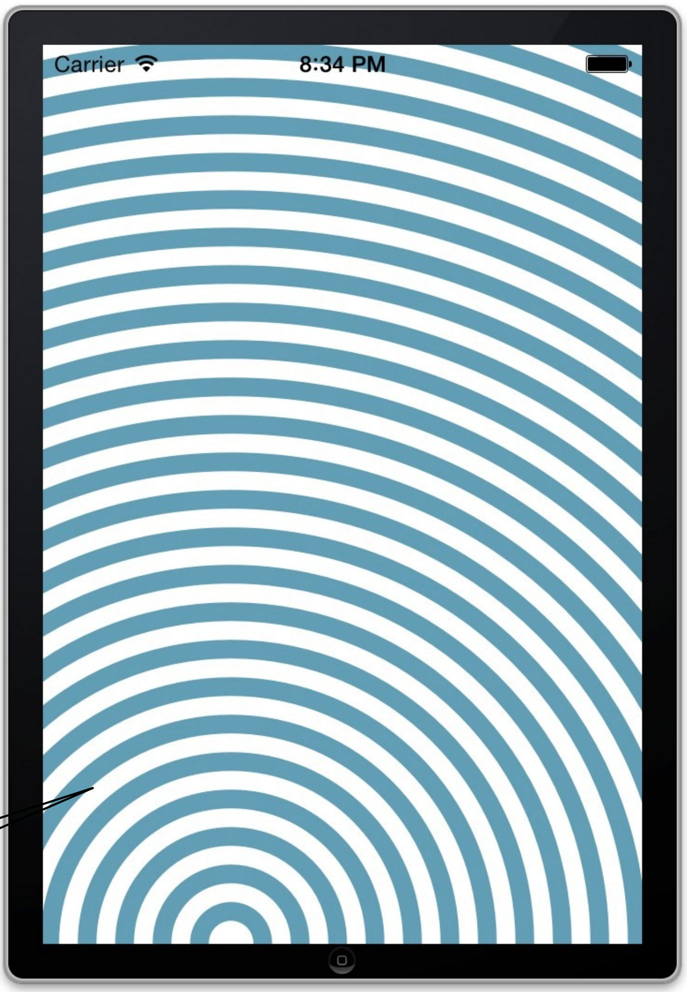
# Scrolling around the Hypnosis view

The World view is twice as big as the screen

The scroll view is as big as the screen



```
BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *

 self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

 // Create CGRects for frames (Twice as big as the main window of the device)
 CGRect screenRect = self.window.bounds;
 CGRect bigRect = screenRect;
 bigRect.size.width *= 2.0;
 bigRect.size.height *= 2.0;

 // Create a screen-sized scroll view and add it to the window
 UIScrollView *scrollView = [[ UIScrollView alloc] initWithFrame: screenRect];
 [self. window addSubview: scrollView];

 // Create a super-sized hypnosis view and add it to the scroll view
 INIHypnosisView *hypnosisView = [[INIHypnosisView alloc] initWithFrame: bigRect];
 [scrollView addSubview: hypnosisView];

 // Tell the scroll view how big its content area (the world view) is
 scrollView.contentSize = bigRect.size;


 self.window.backgroundColor = [UIColor whiteColor];
 [self.window makeKeyAndVisible];
 return YES;
```



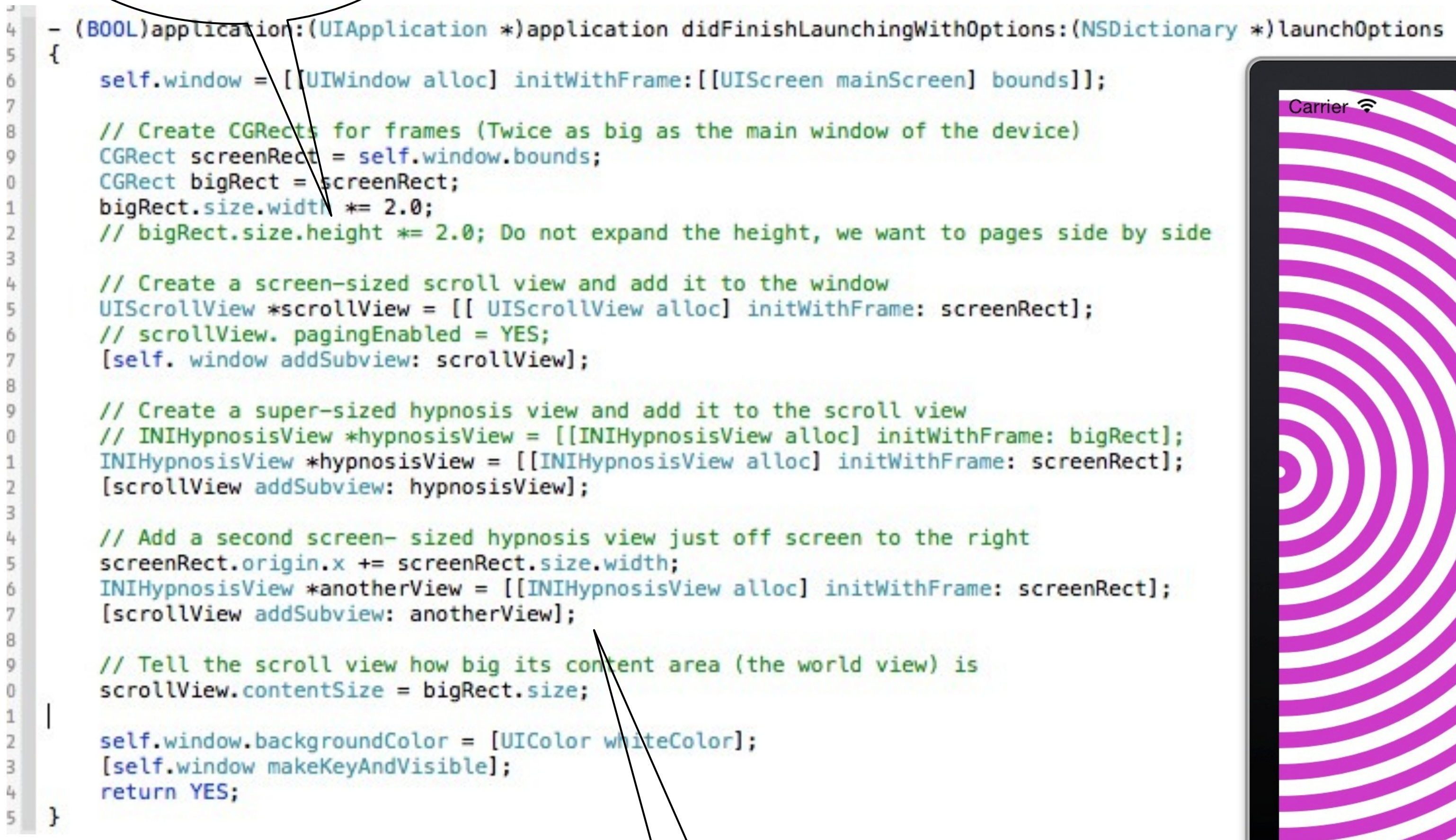
The Hypnosis View is twice as big as the screen

## Panning and Paging

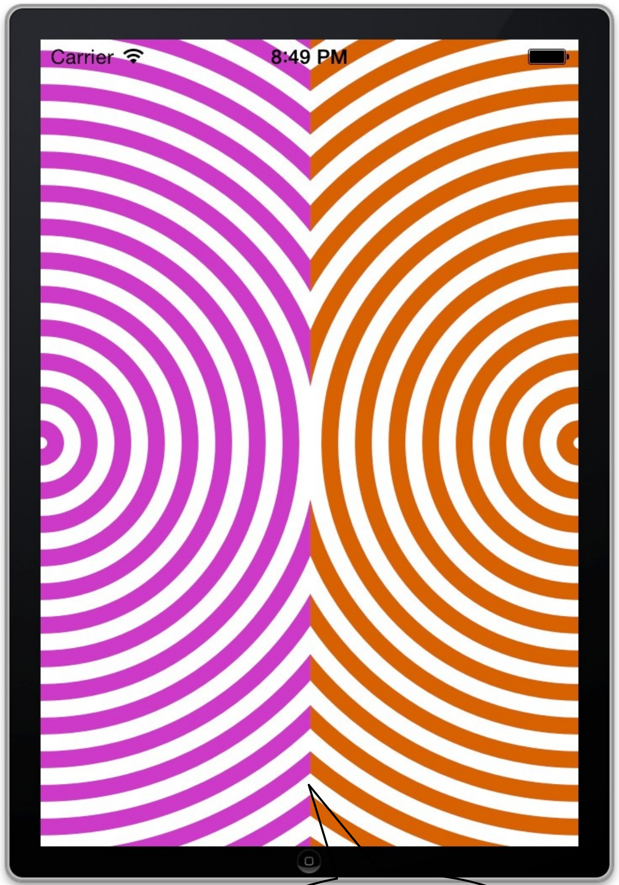
- Recall scrolling is when we moved the peering view around in order to see parts of a much bigger world view.
- Panning and paging is when we have several views of the same size as our peering view. We arrange these views in a grid or adjacent one to the other then we use the UIScrollView to move from on pan to the next pan.
- If we enable paging then the scroll view will snap on the page (pan) boundaries.

# Panning from one View to the next

Enable paging



```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

    // Create CGRects for frames (Twice as big as the main window of the device)
    CGRect screenRect = self.window.bounds;
    CGRect bigRect = screenRect;
    bigRect.size.width *= 2.0;
    // bigRect.size.height *= 2.0; Do not expand the height, we want to pages side by side

    // Create a screen-sized scroll view and add it to the window
    UIScrollView *scrollView = [[ UIScrollView alloc] initWithFrame: screenRect];
    // scrollView. pagingEnabled = YES;
    [self. window addSubview: scrollView];

    // Create a super-sized hypnosis view and add it to the scroll view
    // INIHypnosisView *hypnosisView = [[INIHypnosisView alloc] initWithFrame: bigRect];
    INIHypnosisView *hypnosisView = [[INIHypnosisView alloc] initWithFrame: screenRect];
    [scrollView addSubview: hypnosisView];

    // Add a second screen- sized hypnosis view just off screen to the right
    screenRect.origin.x += screenRect.size.width;
    INIHypnosisView *anotherView = [[INIHypnosisView alloc] initWithFrame: screenRect];
    [scrollView addSubview: anotherView];

    // Tell the scroll view how big its content area (the world view) is
    scrollView.contentSize = bigRect.size;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```



Page Boundary

Create two
hypnosis view next to each other.
Each is a page