Today

Views
  Custom Drawing

Demo
  FaceView
Views

**A view (i.e. UIView subclass) represents a rectangular area**
- Defines a coordinate space
- For drawing
- And for handling touch events

**Hierarchical**
- A view has only one superview \( \ldots \text{var superview: UIView?} \)
- But it can have many (or zero) subviews \( \ldots \text{var subviews: [UIView]} \)
- The order in the subviews array matters: those later in the array are on top of those earlier
- A view can clip its subviews to its own bounds or not (the default is not to)

**UIWindow**
- The UIView at the very, very top of the view hierarchy (even includes status bar)
- Usually only one UIWindow in an entire iOS application \( \ldots \) it's all about views, not windows
Views

- The hierarchy is most often constructed in Xcode graphically
  Even custom views are usually added to the view hierarchy using Xcode

- But it can be done in code as well
  ```
  func addSubview(_ view: UIView) // sent to view's (soon to be) superview
  func removeFromSuperview()     // sent to the view you want to remove (not its superview)
  ```

- Where does the view hierarchy start?
  The top of the (useable) view hierarchy is the Controller's `var view: UIView`. This simple property is a very important thing to understand!
  This `view` is the one whose bounds will change on rotation, for example.
  This view is likely the one you will programmatically add subviews to (if you ever do that).
  All of your MVC's View's UIViews will have this view as an ancestor.
  It's automatically hooked up for you when you create an MVC in Xcode.
Initializing a UIView

As always, try to avoid an initializer if possible

But having one in UIView is slightly more common than having a UIViewController initializer

A UIView's initializer is different if it comes out of a storyboard

init(frame: CGRect) // initializer if the UIView is created in code
init(coder: NSCoder) // initializer if the UIView comes out of a storyboard

If you need an initializer, implement them both ...

func setup() { ... }

override init(frame: CGRect) { // a designated initializer
    super.init(frame: frame)
    setup()
}
required init(coder aDecoder: NSCoder) { // a required initializer
    super.init(coder: aDecoder)
    setup()
}
Initializing a UIView

Another alternative to initializers in UIView ...

awakeFromNib() // this is only called if the UIView came out of a storyboard
This is not an initializer (it’s called immediately after initialization is complete)
All objects that inherit from NSObject in a storyboard are sent this
Order is not guaranteed, so you cannot message any other objects in the storyboard here
Coordinate System Data Structures

**CGFloat**
Always use this instead of Double or Float for anything to do with a UIView’s coordinate system.
You can convert to/from a Double or Float using initializers, e.g.,
let cgf = CGFloat(aDouble)

**CGPoint**
Simply a struct with two CGFloats in it: x and y.
var point = CGPoint(x: 37.0, y: 55.2)
point.y -= 30
point.x += 20.0

**CGSize**
Also a struct with two CGFloats in it: width and height.
var size = CGSize(width: 100.0, height: 50.0)
size.width += 42.5
size.height += 75
Coordinate System Data Structures

**CGRect**

A struct with a CGPoint and a CGSize in it ...

```swift
struct CGRect {
    var origin: CGPoint
    var size: CGSize
}
```

```swift
let rect = CGRect(origin: aCGPoint, size: aCGSize) // there are other inits as well
```

Lots of convenient properties and functions on CGRect like ...

```swift
var minX: CGFloat // left edge
var minY: CGFloat // midpoint vertically
```

```swift
intersects(CGRect) -> Bool // does this CGRect intersect this other one?
intersect(CGRect) // clip the CGRect to the intersection with the other one
contains(CGPoint) -> Bool // does the CGRect contain the given CGPoint?
```

... and many more (make yourself a CGRect and type . after it to see more)
View Coordinate System

- **Origin is upper left**
- **Units are points, not pixels**
  - Pixels are the minimum-sized unit of drawing your device is capable of
  - Points are the units in the coordinate system
  - Most of the time there are 2 pixels per point, but it could be only 1 or even 3
  - How many pixels per point are there? UIView’s `var contentScaleFactor: CGFloat`

- **The boundaries of where drawing happens**
  - `var bounds: CGRect` // a view’s internal drawing space’s origin and size
  - This is the rectangle containing the drawing space in its own coordinate system
  - It is up to your view’s implementation to interpret what `bounds.origin` means (often nothing)

- **Where is the UIView?**
  - `var center: CGPoint` // the center of a UIView in its superview’s coordinate system
  - `var frame: CGRect` // the rect containing a UIView in its superview’s coordinate system
bounds vs frame

*Use frame and/or center to position a UIView*

These are never used to draw inside a view's coordinate system
You might think frame.size is always equal to bounds.size, but you'd be wrong …

Views can be rotated (and scaled and translated)

View B's bounds = ((0,0),(200,250))
View B's frame = ((140,65),(320,320))
View B's center = (300,225)

View B's middle in its own coordinates is …
(bounds.midX, bounds.midY) = (100, 125)

Views are rarely rotated, but don’t misuse frame or center anyway by assuming that.
Creating Views

Most often your views are created via your storyboard
Xcode’s Object Palette has a generic UIView you can drag out
After you do that, you must use the Identity Inspector to changes its class to your subclass

On rare occasion, you will create a UIView via code
You can use the frame initializer … let newView = UIView(frame: myViewFrame)
Or you can just use let newView = UIView()(frame will be CGRect.zero)

Example
// assuming this code is in a UIButtonController
let labelRect = CGRect(x: 20, y: 20, width: 100, height: 50)
let label = UILabel(frame: labelRect) // UILabel is a subclass of UIView
label.text = “Hello”
view.addSubview(label)
Custom Views

When would I create my own UIView subclass?
I want to do some custom drawing on screen
I need to handle touch events in a special way (i.e. different than a button or slider does)
We’ll talk about handling touch events in a bit. First we’ll focus on drawing.

To draw, just create a UIView subclass and override draw(CGRect)
override func draw(_ rect: CGRect)
You can draw outside the rect, but it’s never required to do so.
The rect is purely an optimization.
It is our UIView’s bounds that describe the entire drawing area (the rect is a subarea).

NEVER call draw(CGRect)!! EVER! Or else!
Instead, if you view needs to be redrawn, let the system know that by calling ...
setNeedsDisplay()
setNeedsDisplay(_ rect: CGRect) // rect is the area that needs to be redrawn
iOS will then call your draw(CGRect) at an appropriate time
Custom Views

So how do I implement my `draw(CGRect)`?
You can either get a drawing context and tell it what to draw, or ...
You can create a path of drawing using `UIBezierPath` class (which is how we’ll do it)

Core Graphics Concepts
1. You get a context to draw into (other contexts include printing, off-screen buffer, etc.)
   The function `UIGraphicsGetCurrentContext()` gives a context you can use in `draw(CGRect)`
2. Create paths (out of lines, arcs, etc.)
3. Set drawing attributes like colors, fonts, textures, linewidths, linecaps, etc.
4. Stroke or fill the above-created paths with the given attributes

`UIBezierPath`
Same as above, but captures all the drawing with a `UIBezierPath` instance
`UIBezierPath` automatically draws in the “current” context (`draw(CGRect)` sets this up for you)
`UIBezierPath` has methods to draw (lineto, arcs, etc.) and set attributes (linewidth, etc.)
Use `UIColor` to set stroke and fill colors
`UIBezierPath` has methods to stroke and/or fill
Defining a Path

Create a UIBezierPath

```swift
let path = UIBezierPath()
```

Move around, add lines or arcs to the path

```swift
path.move(to: CGPoint(80, 50))
path.addLine(to: CGPoint(140, 150))
path.addLine(to: CGPoint(10, 150))
```

Close the path (if you want)

```swift
path.close()
```

Now that you have a path, set attributes and stroke/fill

```swift
UIColor.green.setFill()  // note `setFill` is a method in UIColor, not UIBezierPath
UIColor.red.setStroke()  // note `setStroke` is a method in UIColor, not UIBezierPath
path.lineWidth = 3.0    // `lineWidth` is a property in UIBezierPath, not UIColor
path.fill()             // `fill` is a method in UIBezierPath
path.stroke()           // `stroke` method in UIBezierPath
```
Drawing

You can also draw common shapes with UIBezierPath

```swift
let roundRect = UIBezierPath(roundedRect: CGRect, cornerRadius: CGFloat)
let oval = UIBezierPath(ovalIn: CGRect)
... and others
```

Clipping your drawing to a UIBezierPath’s path

```swift
addClip()
```
For example, you could clip to a rounded rect to enforce the edges of a playing card

Hit detection

```swift
func contains(_ point: CGPoint) -> Bool // returns whether the point is inside the path
The path must be closed. The winding rule can be set with usesEvenOddFillRule property.
```

Etc.
Lots of other stuff. Check out the documentation.
UIColor

Colors are set using UIColor

There are type (aka static) vars for standard colors, e.g. let green = UIColor.green
You can also create them from RGB, HSB or even a pattern (using UIImage)

Background color of a UIView

var backgroundColor: UIColor // we used this for our Calculator buttons

Colors can have alpha (transparency)

let semitransparentYellow = UIColor.yellow.withAlphaComponent(0.5)
Alpha is between 0.0 (fully transparent) and 1.0 (fully opaque)

If you want to draw in your view with transparency ...

You must let the system know by setting the UIView var opaque = false

You can make your entire UIView transparent ...

var alpha: CGFloat
View Transparency

What happens when views overlap and have transparency?

As mentioned before, subviews list order determines who is in front. Lower ones (earlier in the array) can “show through” transparent views on top of them. Transparency is not cheap, by the way, so use it wisely.

Completely hiding a view without removing it from hierarchy

```swift
var hidden: Bool
```

A hidden view will draw nothing on screen and get no events either. Not as uncommon as you might think to temporarily hide a view.
Drawing Text

Usually we use a UILabel to put text on screen
But there are certainly occasions where we want to draw text in our draw(CGRect)

To draw in draw(CGRect), use NSAttributedString

```
let text = NSAttributedString(string: "hello")
text.draw(at: aCGPoint)
let textSize: CGSize = text.size // how much space the string will take up
```

Mutability is done with NSMutableAttributedString

It is not like String (i.e. where let means immutable and var means mutable)
You use a different class if you want to make a mutable attributed string ...
```
let mutableText = NSMutableAttributedString(string: "some string")
```

NSAttributedString is not a String, nor an NSString
You can get its contents as an String/NSString with its string or mutableString property
Attributed String

Setting attributes on an attributed string

```swift
func setAttributes(_ attributes: [String:Any]?, range: NSRange)
func addAttributes(_ attributes: [String:Any]?, range: NSRange)
```

Warning! This is a pre-Swift API. NSRange is not a Range.
And indexing into the attributed string is using Int indexing (not String.Index).
It might be helpful to use String's utf16 var to get a String.UTF16View.
A UTF16View represents the String as a sequence of 16 bit Unicode characters.
The characters in a UTF16View will then “line up” with an attributed string’s characters.
But UTF16View is still indexed by String.Index.
So you’ll need to get the UTF16View's characters and use startIndex with index(offsetBy:).

Attributes

```swift
NSForegroundColorAttributeName : UIColor
NSStrokeWidthAttributeName : CGFloat
NSFontAttributeName : UIFont
```

See the documentation under UIKit for (many) more.
Fonts

Fonts in iOS are very important to get right. They are fundamental to the look and feel of the UI.
Fonts

The absolutely best way to get a font in code
Get preferred font for a given text style (e.g. body, etc.) using this UIFont type method ...
static func preferredFont(forTextStyle: UIFontTextStyle) -> UIFont
Some of the styles (see UIFontDescriptor documentation for more) ...
UIFontTextStyle.headline
  .body
  .footnote

There are also “system fonts”
These appear usually on things like buttons
static func systemFont(ofSize: CGFloat) -> UIFont
static func boldSystemFont(ofSize: CGFloat) -> UIFont
Don’t use these for your user’s content. Use preferred fonts for that.

Other ways to get fonts
Check out UIFont and UIFontDescriptor for more, but you should not need that very often
Drawing Images

There is a UILabel-equivalent for images

```
UILabelView
```
But, again, you might want to draw the image inside your `draw(CGRect) ...`

Creating a UIImage object

```
let image: UIImage? = UIImage(named: "foo") // note that its an Optional
You add foo.jpg to your project in the Assets.xcassets file (we’ve ignored this so far)
Images will have different resolutions for different devices (all managed in Images.xcassets)
```

You can also create one from files in the file system

(But we haven’t talked about getting at files in the file system ... anyway ...)
```
let image: UIImage? = UIImage(contentsOfFile: aString)
let image: UIImage? = UIImage(data: aData) // raw jpg, png, tiff, etc. image data
```

You can even create one by drawing with Core Graphics

See documentation for UIViewGraphicsBeginImageContext(CGSize)
Drawing Images

Once you have a UIImage, you can blast its bits on screen

```swift
let image: UIImage = ...
image.draw(at point: aCGPoint)   // the upper left corner put at aCGPoint
image.draw(in rect: aCGRect)    // scales the image to fit aCGRect
image.drawAsPattern(in rect: aCGRect)   // tiles the image into aCGRect
```
Redraw on bounds change?

- **By default,** when a UIView's bounds changes, there is **no redraw**
  Instead, the “bits” of the existing image are scaled to the new bounds size

- **This is often not what you want ...**
  Luckily, there is a UIView property to control this! It can be set in Xcode too.
  ```swift
  var contentMode: UIViewContentMode
  ```

- **UIViewContentMode**
  Don’t scale the view, just place it somewhere ...
  ```swift
  .left/.right/.top/.bottom/.topRight/.topLeft/.bottomRight/.bottomLeft/.center
  ```
  Scale the “bits” of the view ...
  ```swift
  .scaleToFill/.scaleAspectFill/.scaleAspectFit  // .scaleToFill is the default
  ```
  Redraw by calling `draw(CGRect)` again (costly, but for certain content, better results) ...
  ```swift
  .redraw
  ```