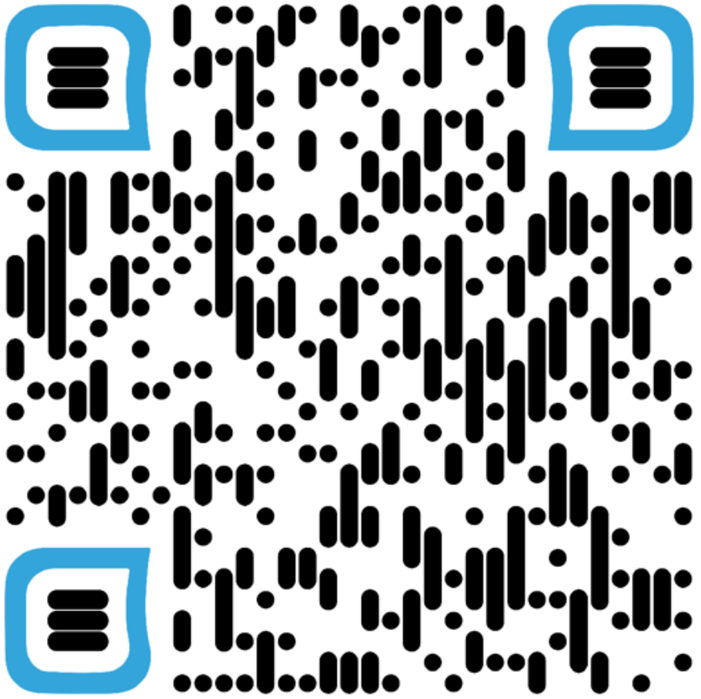
PROF. PEDRO HENRIQUE

DESENVOLVIMENTO PARA IOS 11 COM SWIFT 4

GITHUB DA TURMA



HTTPS://GIT.IO/VF50W

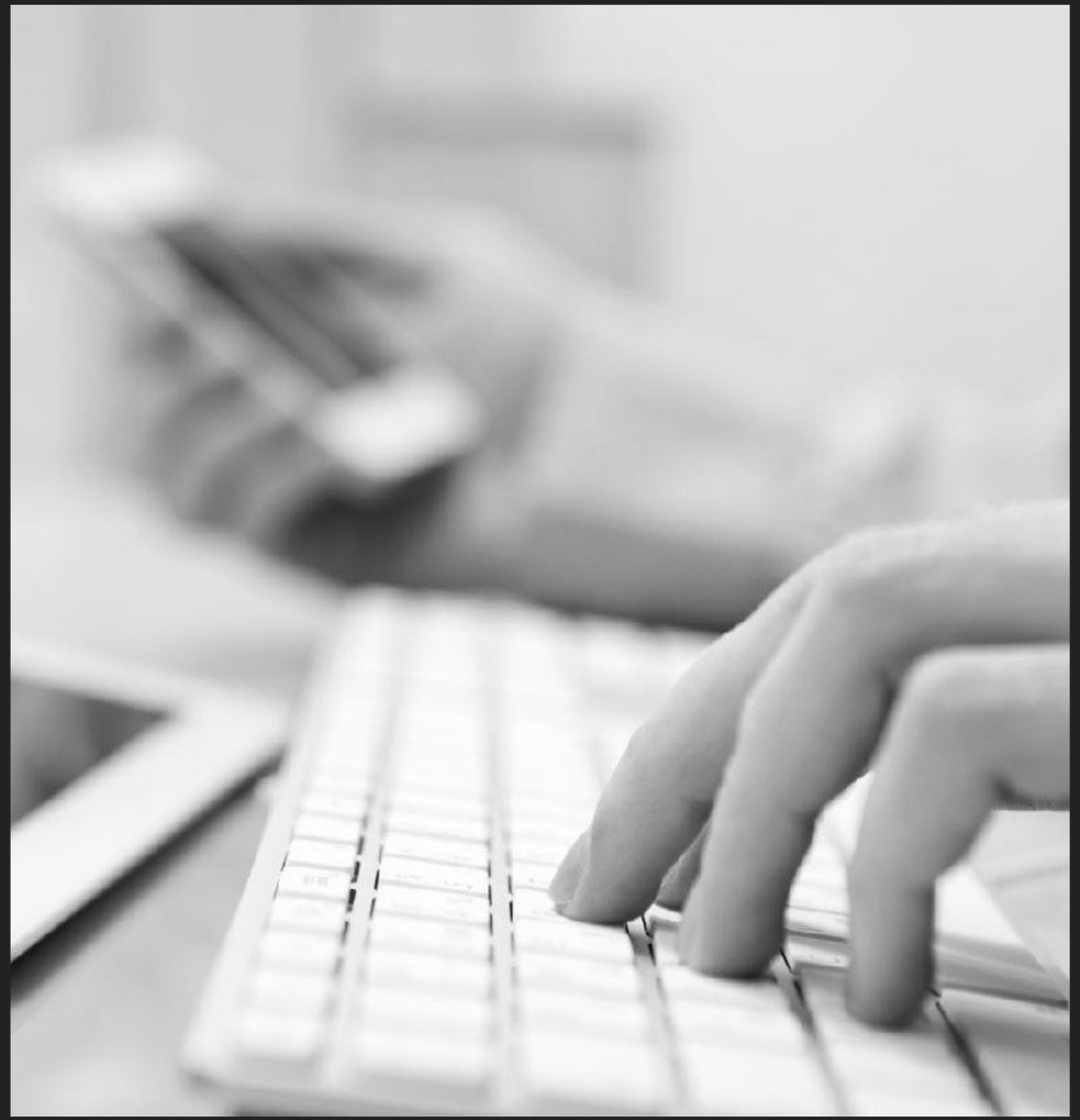
ONDE ENCONTRAR O MATERIAL?

LEIA O QRCODE

ALÉM DO TRADICIONAL BLACKBOARD DO IESB

AGENDA

- Desenhar!
- Programar!



PARA PODERMOS DESENHAR, VAMOS PRECISAR FALAR DE ALGO ANTES...

Views!

VIEWS

- Uma view (**UIView** e suas subclasses) representa uma área retangular;
  - Esta área define um sistema de coordenadas que serve
    - Para desenhar
    - Para escutar a eventos de toque

VIEWS

- São hierárquicas
  - Uma view tem apenas uma superview… var superview: UIView?
  - Pode ter zero ou muitas subviews… var subviews: [UIView]
  - A ordem das subviews no array importa! Aquelas que estão mais perto do final do array estão por cima das que estão mais perto do começo (z-index)
  - Uma view pode limitar o tamanho das subviews para seu próprio tamanho, ou não. O padrão é não.

WINDOWS!!!

- No tipo da hierarquia de views, está a UIWindow. Abaixo dela estão todas as outras views, o que inclui até mesmo a barra de status;
- Em 99,99999% dos casos existe apenas uma UIWindow num aplicativo para iOS;
- Ou seja, aplicativos para iOS são focados em views, não em janelas.

VIEWS

- A hierarquia das views é, na maioria das vezes, construída de forma gráfica através do Xcode
  - Até mesmo views customizadas podem ser adicionadas à hierarquia com a ajuda do Xcode
- Mas também pode ser feito através do código

```
func addSubview(_ view: UIView)
func removeFromSuperview()
```

ONDE A HIERARQUIA COMEÇA?

- O topo da hierarquia utilizável do UIViewController começa com a sua própria view (var view: UIView)
- Essa simples propriedade é algo que é muitíssimo importante entender
- Esta é a view que vai mudar de tamanho quando você girar seu iPhone, por exemplo
- Este é o ponto mais provável para você adicionar subviews de forma programática, se um dia você chegar a fazer isso
- Todas as suas views ligadas a algum MVC vão ter esta view como ancestral
- Ela está automaticamente ligada quando você cria um ViewController no Xcode

INICIALIZANDO UMA UIVIEW

- Como de costume, vamos tentar evitar usar um inicializador
  - Mas o fato de ele existir numa UIView é mais comum do que num UIViewController
- O inicializador é diferente, caso a view venha do Storyboard
  `init(frame: CGRect)` //quando a view é criada programaticamente
  `init(coder: NSCoder)` //quando a view vem do Storyboard
- Se você precisar de um inicializador, o correto é implementar ambos

INICIALIZANDO UMA UIVIEW

- Existe uma alternativa para inicializar uma UIView
- O método **awakeFromNib()** é chamado se e somente se a view vem de um Storyboard ou de um arquivo XIB (falaremos dele no futuro)
- Este método não é um inicializador. Ele é chamado imediatamente após a inicialização ser completada
- Todos os objetos que herdam de NSObject e aparecem no Storyboard recebem uma chamada para este método
- A ordem em que estas chamadas são organizadas não é garantida, então aqui não é o lugar ideal para promover conversas com outros objetos

SISTEMA DE COORDENADAS E ESTRUTURAS DE DADOS RELACIONADAS

- **CGFloat**
  - Sempre usar este tipo no lugar de Double ou Float para qualquer situação que envolver o sistema de coordenadas de uma view
  - Você pode converter de/para Double/Float usando inicializadores
- **CGPoint**
  - Uma struct simples com duas variáveis CGFloat (x e y)
- **CGSize**
  - Uma struct simples com duas variáveis CGFloat (width e height)

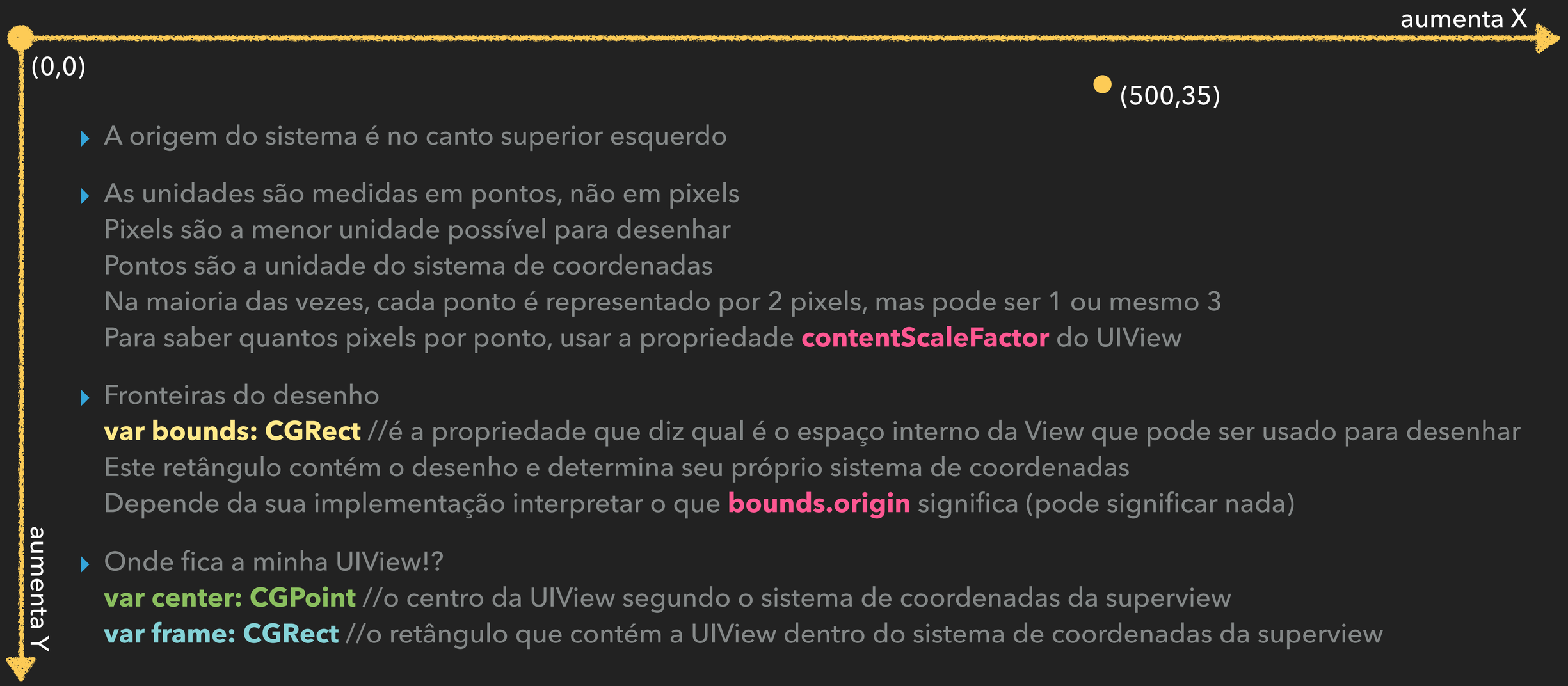
SISTEMA DE COORDENADAS E ESTRUTURAS DE DADOS RELACIONADAS

- **CGRect**
  - Uma struct com um CGPoint e um CGSize dentro…

```
struct CGRect {
    var origin: CGPoint
    var size: CGSize
}
let rect = CGRect(origin: umPonto, size: umTamanho)
//existem também outros inicializadores
```

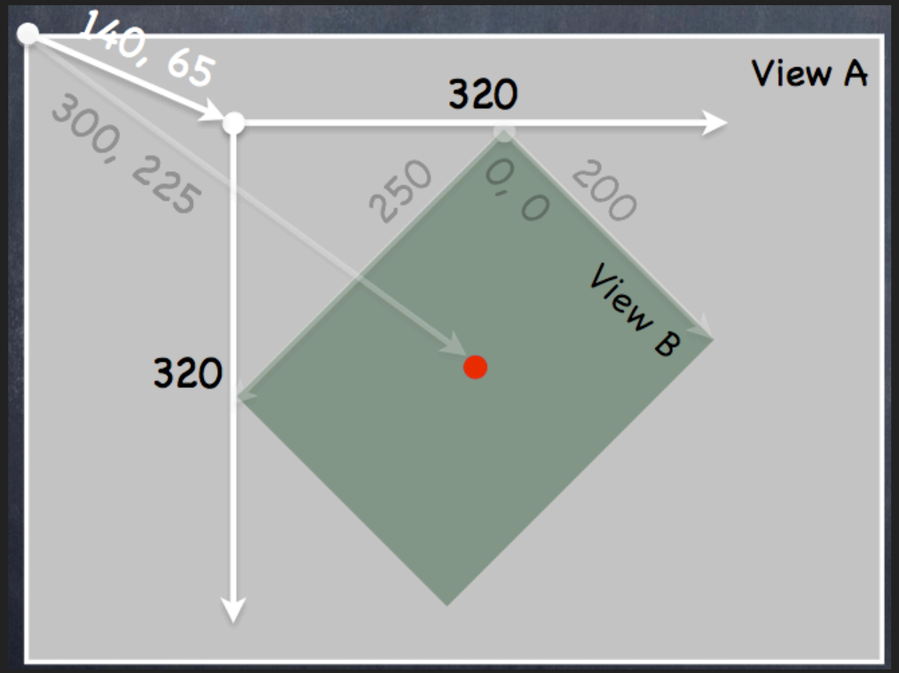
  - Inúmeros métodos utilitários e propriedades interessantes. Veremos na prática.

SISTEMA DE COORDENADAS



- A origem do sistema é no canto superior esquerdo
- As unidades são medidas em pontos, não em pixels
  Pixels são a menor unidade possível para desenhar
  Pontos são a unidade do sistema de coordenadas
  Na maioria das vezes, cada ponto é representado por 2 pixels, mas pode ser 1 ou mesmo 3
  Para saber quantos pixels por ponto, usar a propriedade **contentScaleFactor** do UIView
- Fronteiras do desenho
  var bounds: CGRect //é a propriedade que diz qual é o espaço interno da View que pode ser usado para desenhar
  Este retângulo contém o desenho e determina seu próprio sistema de coordenadas
  Depende da sua implementação interpretar o que **bounds.origin** significa (pode significar nada)
- Onde fica a minha UIView!?
  var center: CGPoint //o centro da UIView segundo o sistema de coordenadas da superview
  var frame: CGRect //o retângulo que contém a UIView dentro do sistema de coordenadas da superview

BOUNDS VS FRAME



- Você pode usar frame e/ou center para posicionar uma UIView
- Embora estes nunca devem ser usados para desenhar dentro do sistema de coordenadas da view
- Se você pensou que frame.size é igual a bounds.size, você errou…
- As views podem ser rotacionadas, escaladas e transladas
- Então…
  b.bounds = ((0,0), (200,250))
  b.frame = ((140,65), (320, 320))
  b.center = (300, 225)
  O centro de B no seu próprio sistema de coordenadas é:
  (bounds.midX, bounds.midY) == (100,125)
- As views raramente são rotacionadas, mas não faça o uso incorreto do frame ou do center ao assumir isso.

CRIANDO VIEWS

- Na maioria dos casos, as views são criadas com a ajuda do seu Storyboard
  Na paleta de componentes do Xcode existe um componente que representa uma UIView genérica que você pode usar
  Depois que você arrastar esta UIView para sua tela, você precisa usar a aba **Identity Inspector** para mudar a classe que ela representa

- Em raras ocasiões, você vai criar uma UIView puramente via código
  Daí você pode usar o inicializador que recebe o frame...
  `let novaView = UIView(frame: meuFrame)`
  Ou você pode simplesmente usar `let minhaView = UIView()` (o frame vai ser **CGRect.zero**)

VIEWS PERSONALIZADAS

- Quando eu crio minhas próprias views?
  Quando você precisar fazer algum desenho na tela
  Quando você precisar responder à algum evento de toque de algum jeito especial (diferente do que um botão ou um slider faz)
  Vamos falar de eventos de toque depois. Agora vamos focar no desenho

- Para desenhar, simplesmente crie uma classe filha de UIView e sobrescreva o método draw(CGRect)
  override func draw(_ rect: CGRect)
  Você pode desenhar até mesmo fora do rect, mas não é o ideal
  O rect é meramente uma otimização
  O tamanho real da área disponível fica no bounds. O rect é apenas uma subarea

- Você **NUNCA** deve evocar o método draw(CGRect). Em hipótese alguma! **JAMAIS**! Nunca mesmo!
  Se você acredita que sua view precisa ser redesenhada, informe ao iOS através do método **setNeedsDisplay**. O sistema operacional vai, então, evocar o método draw num momento oportuno


quer aprender como faz isso fera ?
vem cmg que eu te ensino
CriarMeme

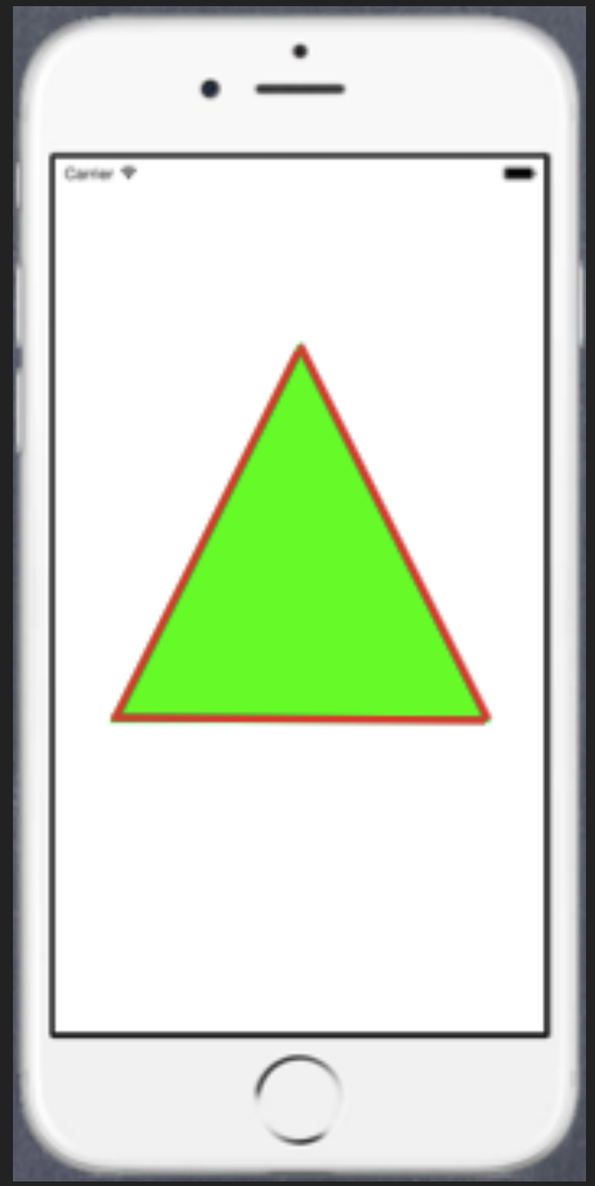
UMA VIEW PARA CHAMAR DE SUA

- Como eu implemento esse método draw?
  Jeito difícil: criar um contexto de desenho e dizer, ponto a ponto, o que desenhar
  Jeito fácil: criar um UIBezierPath e fazer magia negra
- Conceitos do framework CoreGraphics (Jeito difícil)
  1. Obter um contexto para desenhar (existem outros contextos... impressão, desenho fora da tela, buffer, etc). A função UIGraphicsGetCurrentContext() te fornece um contexto para desenhar
  2. Criar paths (à partir de linhas, arcos, etc)
  3. Configurar os atributos do desenho (cores, fontes, texturas, espessuras, etc)
  4. Traçar e/ou preencher o desenho com os dados atributos
- **UIBezierPath** (jeito fácil)
  Mesma coisa do de cima, mas encapsula toda a parte difícil numa instância da classe UIBezierPath. Ele automaticamente já usa o contexto “atual" para desenhar, além de já oferecer métodos para desenhar e configurar atributos. Ele tem também os métodos para traçar e para preencher o desenho com as cores indicadas.


I THOUGHT...
FINGER PAINTING WAS FOR CHILDREN!
memecrunch.com

DEFININDO UM PATH

- Crie um UIBezierPath
  ```
  let path = UIBezierPath()
  ```
- Movimente, adicione linhas ou arcos
  ```
  path.move(to: CGPoint(80,50))
  path.addLine(to: CGPoint(140,150))
  path.addLine(to: CGPoint(10,150))
  ```
- Feche o path (caso queira)
  ```
  path.close()
  ```
- Configure os atributos
  ```
  UIColor.green.setFill()
  UIColor.red.setStroke()
  path.linewidth = 3.0
  path.fill()
  path.stroke()
  ```



DESENHANDO...

- Você também pode desenhar formas comuns com o UIBezierPath
  let arredondado = UIBezierPath(roundedRect: CGRect, cornerRadius: CGFloat)
  let oval = UIBezierPath(ovalIn: CGRect)
  e muitos outros...
- Clip
  Você pode limitar o próximo desenho ao interior do desenho anterior com o método **addClip()**
- O método contains(_ point: CGPoint) retorna true ou false para dizer se o ponto está dentro do seu path, que deve estar fechado, obviamente

CORES

- As cores são obtidas através de instâncias de UIColor
  Existem constantes de tipo para as cores padrão (ex.: UIColor.green)
  Você pode criar suas próprias cores usando RGB, HSB ou ainda usando uma textura (com um UIImage)
- Toda UIView tem uma cor de fundo que é descrita na variável backgroundColor
- As cores podem ter um canal alpha (transparência) onde alpha 0.0 significa totalmente transparente e 1.0 significa totalmente opaco
  let amareloTransparente = UIColor.yellow.withAlphaComponent(0.5)
- Se você quer desenhar com transparência, você precisa comunicar o iOS disso, setando false para a variável opaque
- Você ainda pode fazer sua view inteira ficar transparentes, modificando o valor da variável alpha

TRANSPARÊNCIA DAS VIEWS

- O que acontece quando as views se sobrepõe e são transparentes?
  Conforme já foi dito, a ordem da lista de **subviews** determina quem fica na frente de quem. As views inferiores (primeiras posições do array) podem ser vistas através das views transparentes que estivem acima. Atenção!!! Transparência não é um recurso computacional barato, então use com prudência.

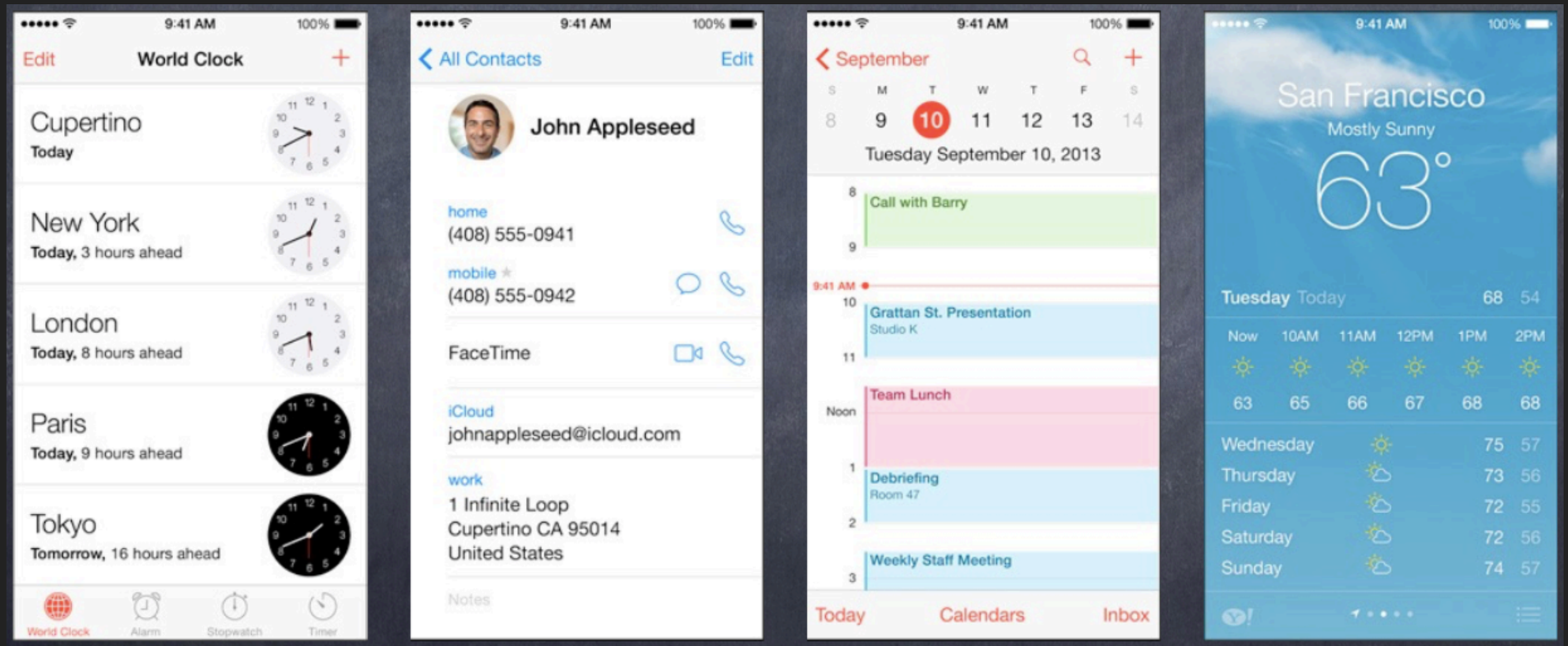
- É possível ainda esconder completamente uma view sem removê-la da hierarquia com a variável **hidden**
  Uma view escondida não desenha nada na tela e também não pode receber eventos. Usar esta técnica é bem mais comum do que vocês imaginam.

DESENHANDO TEXTO

- Normalmente usamos um UILabel para colocar texto na tela
  Mas em certas ocasiões vamos querer desenhar texto no método draw
- Para fazer isso, usamos um objeto da classe NSAttributedString
  let text = NSAttributedString(string: “hello”)
  text.draw(at: umCGPoint)
  let textSize: CGSize = text.size //quanto espaço a string vai ocupar
- Mutabilidade é feita com a classe NSMutableAttributedString
- NSAttributedString não é a mesma coisa que String ou NSString

FONTES

- É muito importante saber escolher as fontes certas no iOS. Elas são fundamentais para compor o look n'feel da sua interface



FONTES

- A melhor forma de obter uma fonte no código é obter a fonte preferencial para um dado estilo de texto (body, head, footnote, etc) através do método **static preferredFont(forTextStyle: UIFontTextStyle) -> UIFont**
- Existem ainda as "fontes do sistema”, que aparecem em coisas como botões, por exemplo. Você deve evitar o uso destas em texto que seja conteúdo. Use a forma acima para o conteúdo.
- Outras formas de obter fontes você pode conferir na documentação para UIFont e UIFontDescriptor, mas estes não devem ser tão necessários

DESENHANDO IMAGENS

- Existe um equivalente ao UILabel para imagens, o UIImageView, mas novamente você pode querer desenhar uma imagem dentro do seu método draw...
- Para desenhar uma imagem, você vai precisar de uma
  `let img: UIImage? = UIImage(named: “minhaFoto")`
  Vamos falar de como adicionar imagens ao projeto daqui a pouco
- Você pode ainda criar imagens à partir do filesystem (mas nós nem falamos de filesystem ainda, então essa fica para depois)
- Você pode ainda desenhar uma imagem com o CoreGraphics. Veja a documentação para UIGraphicsBeginImageContext(CGSize)

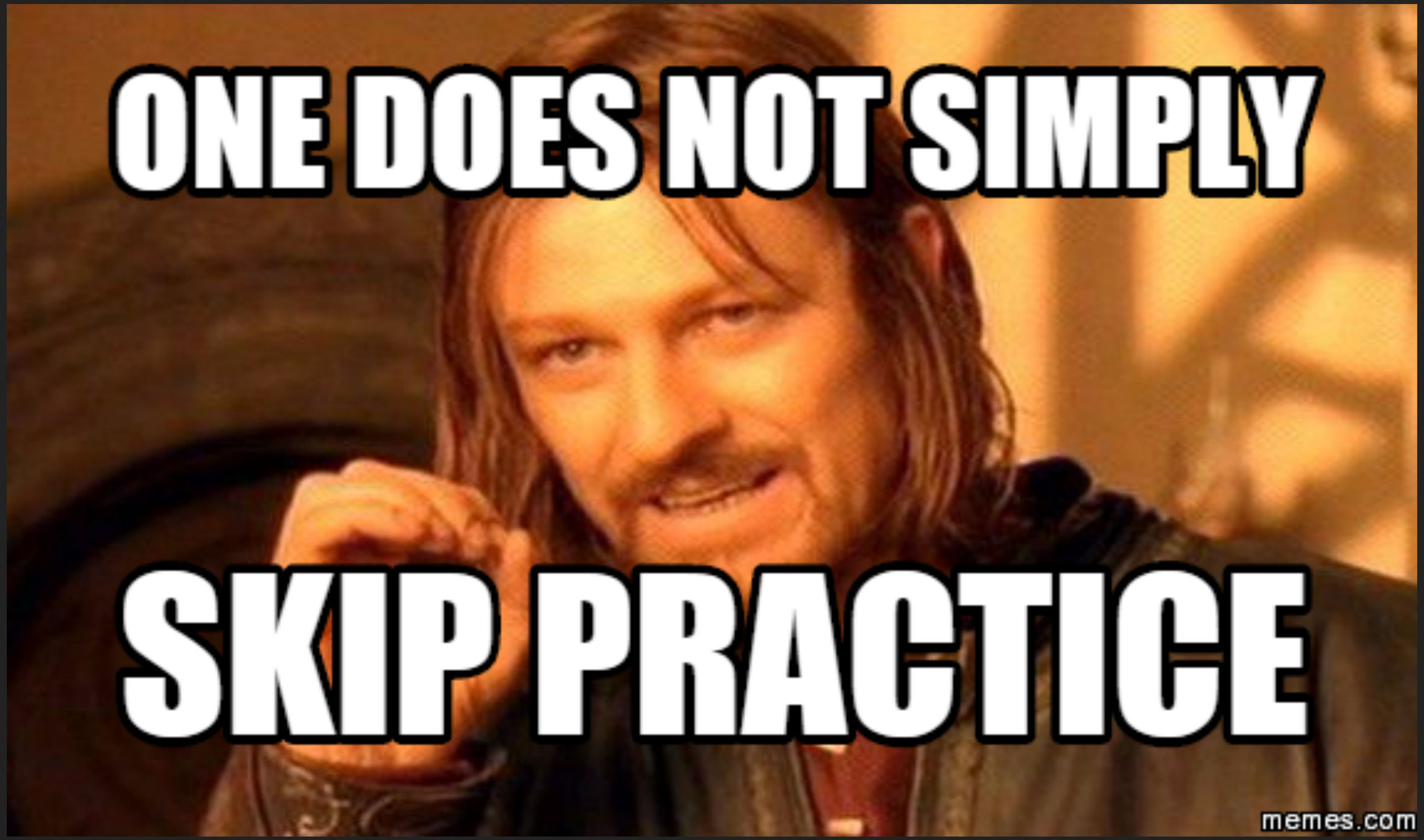
DESENHANDO IMAGENS

- Uma vez que você tem sua UIImage em mãos, você pode colocá-la para desfilar na tela:

```
let image = UIImage ...
image.draw(at point: umPonto) //alinha o canto superior esquerdo da imagem com o ponto
image.draw(in rect: umRetangulo) //reduz para caber, ignorando a proporção
image.drawAsPattern(in rect: umRetangulo)
```

E QUANDO A TELA MUDA? REDESENHAR???

- Por padrão, quando uma UIView muda de tamanho, não há redesenho.
- Eventualmente, este não é o comportamento que você deseja
- Por sorte, a UIView tem uma variável que permite controlar isso e ela pode ser modificada via código e/ou via Xcode.
  var contentMode: UIViewContentMode
- Valores de UIViewContentMode
  Não escalar... apenas reposicionar: left/right/top/bottom/topRight/topLeft/bottomRight/bottomLeft/center
  Escalar: scaleToFill (padrão) /scaleAspectFill/scaleAspectFit
  Redesenhar: redraw (cuidado! O custo computacional é alto, mas os resultados são excelentes)


ONE DOES NOT SIMPLY
SKIP PRACTICE
memes.com

EXERCÍCIO

- Desenhar um triângulo
- No exemplo que fizemos juntos, adicionar componentes para modificar a carinha (abrir e fechar os olhos, modificar o sorriso). Sugestão: botão e slider

- DESAFIO
  - Desenhar uma estrela, sem que as linhas atravessem por dentro da figura. O número de pontas da estrela deve ser um @IBInspectable.
  - Desenhar uma joaninha

- ATENÇÃO!!! Todos os desenhos devem ser feitos em Views separadas e as propriedades devem ser ajustáveis via Xcode (@IBInspectable e @IBDesignable)