

Intermediate

Core Graphics

Part 2: Transforms

Core Graphics Hands-On Challenges

Copyright © 2016 Razeware LLC.

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written permission of the copyright owner.

This book and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

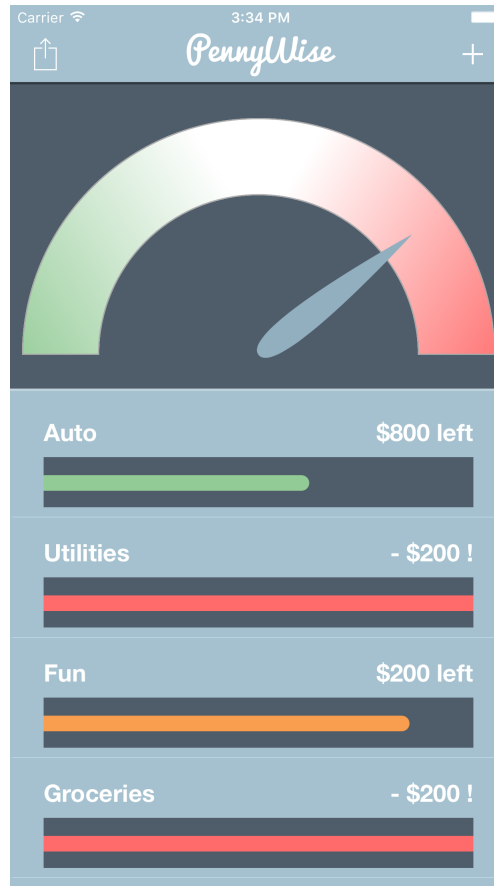
All trademarks and registered trademarks appearing in this book are the property of their respective owners.



Challenge: CALayer Transforms

Transforms are not only for Core Graphics - they're used by views and Core Animation layers too.

In this challenge you're going to rotate the pointer layer for the summary view.



Whenever an expense is made, the pointer will move further around. When your total budget has been spent, the pointer will be all the way to the right.

You're going to set the transform of the pointer to be the correct rotation angle for the amount spent.



MainViewController is the delegate for ExpenseViewController. Whenever an expense is saved, MainViewController executes the delegate method `expenseViewController(_:didExpenseCategory:amount:)`. This saves the expense and calls `calculateBudget()` to update the budget.

`calculateBudget()` then updates SummaryView's `percentSpent` property.

Whenever `percentSpent` changes, you will call a method to rotate the pointer.

In SummaryView, create a method:

```
func rotatePointer() {  
}
```

Change the class property from:

```
var percentSpent:Float = 0
```

to

```
var percentSpent:Float = 0.5 {  
    didSet {  
        rotatePointer()  
    }  
}
```

Now every time that `percentSpent` is updated, `rotatePointer()` will be called.

In `rotatePointer()`, add:

```
if percentSpent > 1 {  
    percentSpent = 1  
}
```

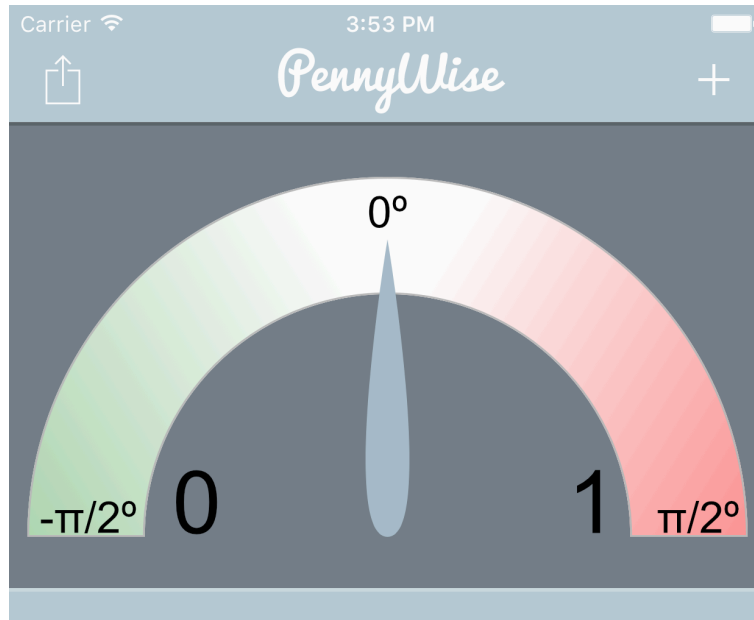
`percentSpent` is a value between 0 and 1. When `percentSpent` is 1, the pointer will point furthest to the right. You never want it to be more than 1 even if the budget has blown out.



Now to work out the rotation.

When the pointer layer is in its default position, the rotation angle is 0° , pointing straight up. When the pointer layer is pointing to the right, it will be 90° , and when pointing to the left, -90° . So when the percent spent value is 0, the rotation angle will be -90° and when the percent spent is 1, the rotation angle will be 90° .

The radian equivalent of 90° is $\pi/2$.



So taking that into account, this is the rotation calculation for the transform.
(Remember, it's option+P to type π .)

```
let transform = CGAffineTransformMakeRotation( $-\pi/2 + \pi * \text{CGFloat}(\text{percentSpent})$ )
```

This creates a CGAffineTransform variable with the calculated rotation angle.

Assign it to the layer's transform:

```
pointerLayer.setAffineTransform(transform)
```

Run the app, try putting in various expenses and see the pointer move. Because layers have built in animation, the rotation of the pointer is even animated for free.

