306: Animating Auto Layout Constraints

Part 4: Challenge Instructions

306: Animating Auto Layout Constraints
Part 4: Challenge Instructions

Push it further

We have two challenges for you to keep experimenting with Auto Layout animations:

1. **Animate the Understanding strip** to try combining "normal" view animations with constraint animations.
2. **Add a new UI element** to the view controller and animate its constraints.

By the time you have finished these challenges, you'll have really learned a solid new skill – Auto Layout animations!

Challenge A: Custom Understanding Animations

This time you will be (almost) on your own – you will add a new custom constraint animation to the image in the Understanding strip. You will combine two different constraint animations and a "normal" animation on the image's `alpha` property.

Since you should posses all skills how to complete this challenge you will receive only very basic instructions how to proceeed about it.

Add a new method called `toggleUnderstanding(tap: UITapGestureRecognizer)` to `ViewController` and change the `understandingTap` in `viewDidLoad()` to invoke that new method. (Keep consulting the existing code throughout the challenge.)

Inside `toggleUnderstanding(...)` call `toggleView(...)` and create an `isSelected` constant just like for the other strips.

Than just like for the reading strip call a method to animate the image:

```
toggleUnderstandingImageViewSize(understandingImage, isSelected:
isSelected)
```

Create the new method as:

```
func toggleUnderstandingImageViewSize(imageView: UIImageView, isSelected:
Bool)
```

Lookup `toggleReadingImageSize(...)` and in `toggleUnderstandingImageViewSize(...)` in the same way create a loop over the `imageView`'s parent's constraints.

Add an `if` condition to find the `.Height` constraint. When you find that constraint – remove it and create a new `.Height` constraint between the `imageView` (as the first item) and its parent view.

For the multiplier of the new constraint use `1.6` if the view is selected (use the `isSelected` method parameter) and `0.45` if not.

Now still inside the for loop body add a second `if`. This time check for a constraint where `firstItem` is the `imageView` (as before) but the `firstAttribute` is `.CenterY`. This constraint is the one aligning the image vertically.

Inside the `if` body remove the existing constraint. Then replace it with a new one:

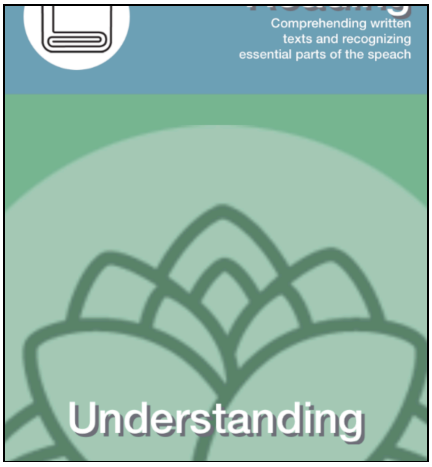
```
let newConstraint = NSLayoutConstraint(
  item: constraint.firstItem,
  attribute: .CenterY,
  relatedBy: .Equal,
  toItem: (constraint.firstItem as UIView).superview!,
  attribute: .CenterY,
  multiplier: isSelected ? 1.8 : 0.75,
  constant: 0.0)
```

This new constraint will move the image away when the strip is selected. The two constraints will animate together when you call `layoutIfNeeded()`.

It is actually time to add the animation code – scroll to `toggleUnderstanding(...)` and add:

```
UIView.animateWithDuration(0.5, delay: 0.0, options:
.CurveEaseOut, animations: {
  self.understandingImage.alpha = isSelected ? 0.33 : 1.0
  self.understandingView.layoutIfNeeded()
}, completion: nil)
```

This code will decrease the alpha of the image when the strip is selected and animate the layout changes. Run the project and tap the strip at the bottom – you will see all three animations execute simultaneously:



Now wrap up by setting `deselectCurrentView` at the bottom of `toggleUnderstanding(...)` – you need to call `toggleUnderstandingImageViewSize(...)` and reset the alpha property of `understandingImage`. This should be walk in the park for you after everything you achieved in this tutorial – I'll see in you in the next challenge.

Challenge B: Add a button and animate its constraints

So far you have been working on the fictional Languages app main menu – but you didn't provide the user with any way to start the learning process. That's why for this challenge you will add dynamically a button saying "Start excercise" to the Understanding strip when the user taps on it.

You will use all of your new skills to create the required animations and to dynamically create the correct constraints to first position the button and then animate it on its place.

At the bottom of `toggleUnderstanding(...)` add:

```
showUnderstandingButton(isSelected)
```

This should call yet another helper method and show or hide the button depending on whether the strip is currently selectd or not.

Before you can add the showUnderstandingButton(...) you need a couple of helper declarations first. Add a new class property:

```
var understandButton: UIButton!
```

You will use this property to keep hold of the currently displayed button.

Next you will need a helper method to hide the button:

```
func hideUnderstandingButton() {
  UIView.animateWithDuration(0.5, delay: 0.0, options:
.CurveEaseOut | .BeginFromCurrentState, animations: {
    self.understandButton.alpha = 0.0
  }, completion: {_ in
    self.understandButton.removeFromSuperview()
  })
}
```

Now you can add the initial version of `showUnderstandingButton(...)`:

```
func showUnderstandingButton(isSelected: Bool) {
```

```
    if !isSelected {
      hideUnderstandingButton()
      return
    }

  }
```

This code will call hideUnderstandingButton() if the strip is already selected – a good start!

Next (this part of the code will execute only when the strip is being currently selected) you will create the button by code. Append to the method body:

```
understandButton = UIButton.buttonWithType(.Custom) as UIButton
understandButton.backgroundColor = UIColor(red: 1.0, green: 1.0,
blue: 1.0, alpha: 0.9)
understandButton.setTitleColor(UIColor.blackColor(), forState:
.Normal)
understandButton.setTitle("Start excercise", forState: .Normal)
understandButton.layer.cornerRadius = 10.0
understandButton.layer.masksToBounds = true
understandButton.setTranslatesAutoresizingMaskIntoConstraints(fals
e)
understandingView.addSubview(understandButton)
```

You adjust all button properties to make it fit nicely in the UI and you add it to `understandingView`.

Now comes the time to create all the constraints you need in order to position the button centered horizontally and just out of the top edge of `understandingView`. You will animate it from this position into the view.

Add to the method the code to create four constraints. They all have item of `understandingButton`, and `relatedBy` parameter equal to `.Equal`. For the rest consult the table below:

Name	attribute	toItem	attribute(2nd)	multiplier	constant
conX	.CenterX	understandingView	.CenterX	1.0	0.0
conY	.Bottom	understandingView	.Bottom	0.75	-understandingView. frame.size. height
conWidth	.Width	understandingView	.Width	0.5	0.0
conHeight	.Height	understandButton	.Width	0.25	0.0

This certainly was a lot of code eh? But you're very close to wrapping everything up.

After you created all the constraints make a call to `layoutIfNeeded()` to force Auto Layout to position the button at the correct location. Add this code to do that:

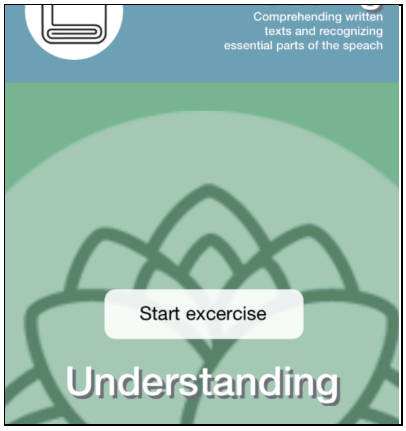
```
understandingView.layoutIfNeeded()
```

Now the only part left is to add the animation code to reset the intentionally offsetted Y coordinate of the button in order to get it to animate to the correct location inside the Understanding view.

Add:

```
UIView.animateWithDuration(1.33, delay: 0.0,
usingSpringWithDamping: 0.8, initialSpringVelocity: 0.0, options:
.BeginFromCurrentState, animations: {
  conY.constant = 0.0
  self.understandingView.layoutIfNeeded()
}, completion: nil)
```

This should trigger a new layout pass but this animated; when you run the project you will see the button appear like so:



If you get this part working, congrats – not only do you have great experience with Auto Layout, but you already have bent it to your will and are an Auto Layout animations master:]

If you would like to – you can continue playing with the project, add more buttons, show more stats ... the sky is the limit!