Animations

What is an animation?

An animation is the time difference between two states of an object. Ex:

-Moving a label from point A to B in 5 seconds

-Changing an images alpha from 0 to 1 in 1 second

-rotating a button 360 degrees in 2 seconds

Why use interface animations

- It can help users identify connections between interface elements and views
- Give feedback to users so they know whats happening
- Show the hierarchy of pages and screens
- Attract attention to important elements and functions
- Improves the user experience
- Makes you app memorable

Click here to read more about interface animations

Simplest animation function

func animate(withDuration duration: TimeInterval, animations: () -> Void)

This method takes two arguments: a duration of type **TimeInterval** (which is an alias for a **Double**) and an animations variable that is a closure.

Closure refresher

A closure is basically an anonymous function, a function without a name. One instance of where closures are useful is when a function/method has a closure as its argument. Remember functions/methods are types of closures.

```
{ (arguments) -> return type in

// code

}
```

You write a closure expression inside braces ({}). The closure's arguments are listed inside parentheses immediately after the opening brace. A closure's return type comes after the parameters and uses the regular syntax. The keyword in is used to separate the closure's arguments and return type from the statements inside of its body.

animations

func animate(withDuration duration: TimeInterval, animations: () -> Void)

In the animations closure you will be defining what the end state of the object(s) you are trying to manipulate. They will go from their original state to the new state in TimeInterval amount of time

Animation completion

The method **animate(withDuration:animations:)** returns immediately. That is, it starts the animation, but it does not wait for the animation to complete. What if you want to know when an animation completes? For instance, you might want to chain animations together or update another object when the animation completes. To know when the animation finishes, pass a closure for the completion argument

UIView.animate(withDuration: timeInterval, delay: TimeInterval, options: UIView.AnimationOptions = [ ] , animations: ( ) -> Void, completion: (Bool) -> void? = nil)

Will be the new format we use visit apple docs for more information

Completion

```
UIView.animate(
        withDuration: timeInterval,
        delay: TimeInterval,
        options: UIView.AnimationOptions = [ ] ,
        animations: ( ) -> Void,
        completion: (Bool) -> void? = nil )
```

This is the form you will usually see it in. ignore the options argument for now, delay will set a time for when the animation should start.
Completions is the focus of this slide, acces a closure statement that will execute when the animation is complete.

This is extremely useful if you want to chain several animation one after the other, or if you need to reset some objects, hidden or visual

Animating Constraints

Not only can view objects be animated so can constraints. If an view object is tied to a constraint and you move said constraint it will also move the object.

Before animating an object you will need to set up an IBOutlet to that constraint.

Make sure it is of type NSLayoutConstraint!

Any time you change the value of a constraint constant you will have to use the view.layoutIfNeeded() method within the animation argument to tell the system to animate this constraint change. This is because after a constraint is modified, the system needs to recalculate the frames for all of the related views in the hierarchy to accommodate this change. It would be expensive for any constraint change to trigger this automatically.

Timing Functions

The acceleration of the animation is controlled by its timing function. By default, animations use an ease-in/ease-out timing function. To use a driving analogy, this would mean the driver accelerates smoothly from rest to a constant speed and then gradually slows down at the end, coming to rest.

The options parameter takes in a **UIViewAnimationOptions** argument. Why is this argument in square brackets? There are many options for an animation in addition to the timing function. Because of this, you need a way of specifying more than one option – an array. **UIViewAnimationOptions** conforms to the **OptionSet** protocol, which allows you to group multiple values using an array.

Here are some of the possible animation options that you can pass into the options parameter. Animation curve options

Control the acceleration of the animation

Possible values are:

- UIViewAnimationOptions.curveEaseInOut
- UIViewAnimationOptions.curveEaseIn
- UIViewAnimationOptions.curveEaseOut
- UIViewAnimationOptions.curveLinear