iOS Dev Accelerator Week3 Day3

- Animation in iOS
- Custom View Controller Transitions
- Web Tools Wednesday: Express and Routes
- App Idea Presentations

Animation in iOS*

*Sourced from Motion Design for iOS by Mike Rundle

Animating your app

- Thanks to iOS7’s new design principles, there are really only 3 reasons to use animations in your app:
  - Transition: Smoothly animating from one visual state to the next helps the user understand what the app is doing and why.
  - Focus: Direct the user’s attention to a specific aspect of the interface.
  - Delight: Make things look awesome and appealing.

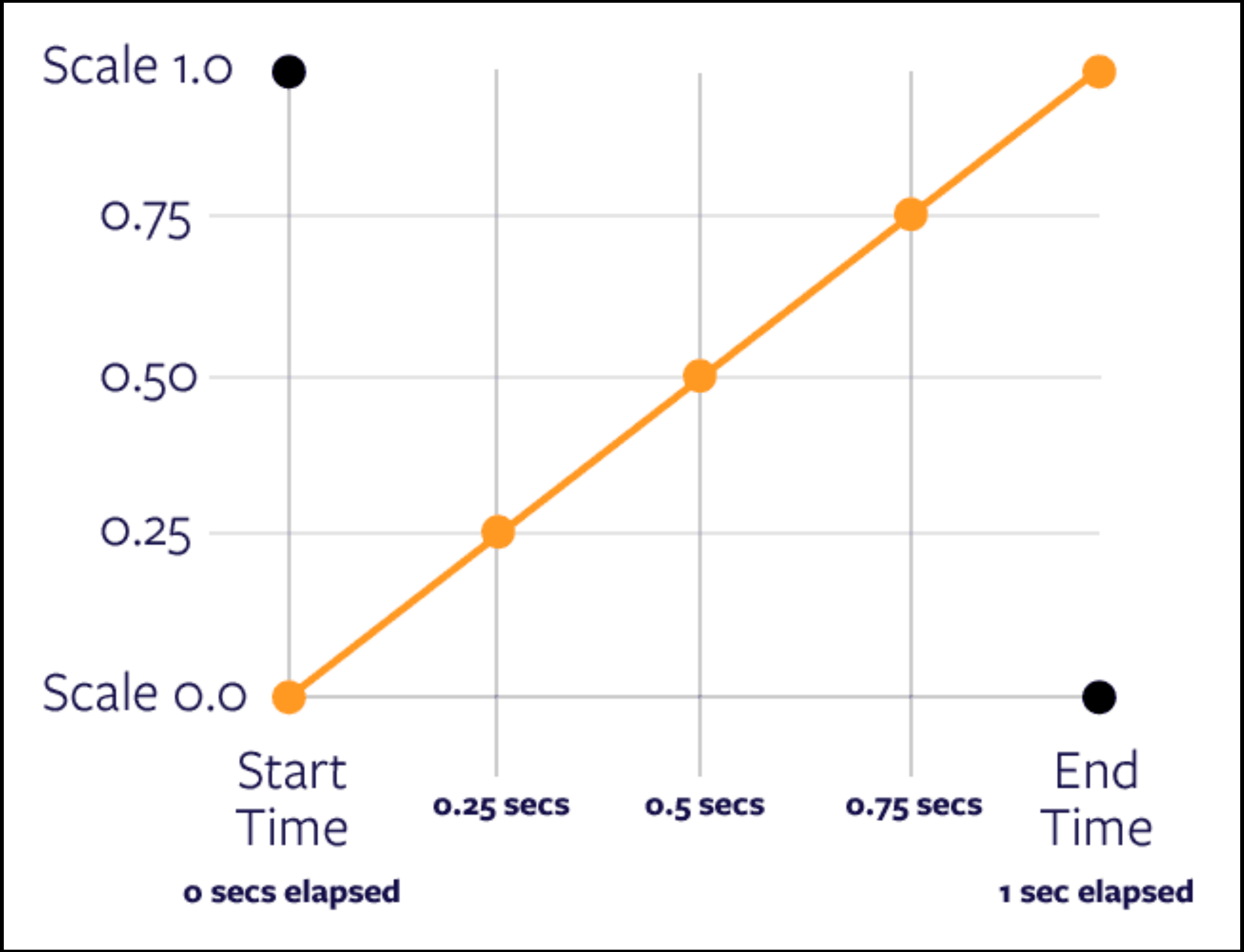
Planning out your animations

1. What are the initial properties of the item?
2. What are the final properties of the item?
3. How long should the animation take?
4. Whats happening to this item while it is animating?
5. What will happen once this item is done animating?

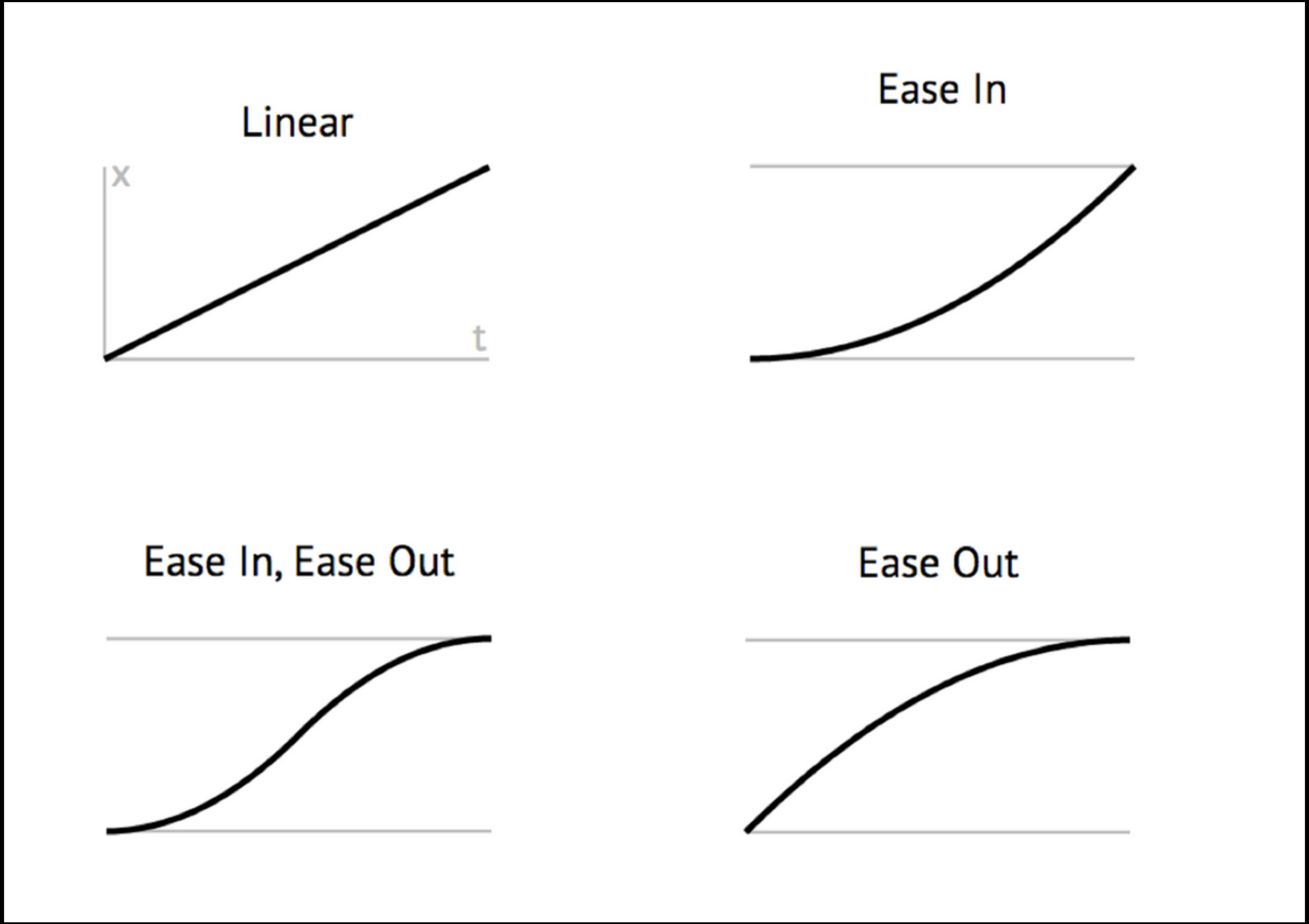
Properties that can be animated

- Position/Size/Frame: Changing the X and Y values of a views origin
- Opacity: Changing the alpha from in the range of 0 to 1
- Scale:Increase or decrease the size of a view. 1 represents its normal size.
- Color: transition one color value to another.
- Rotation: Rotate a view by radians.
- 3D Transform: Rotate the third dimension of a view
- Constraints!

Timing is everything



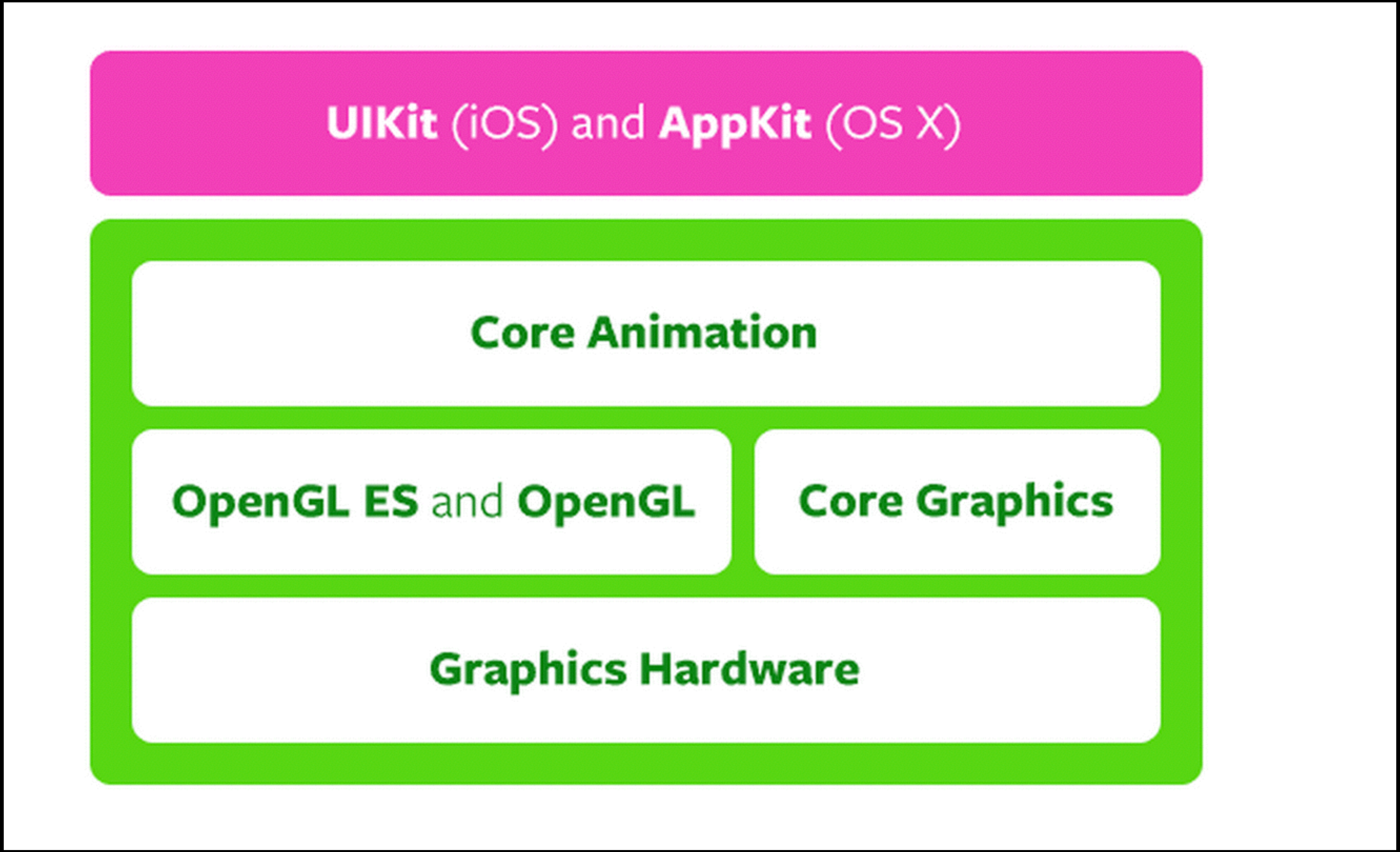
Standard animation curves



Core Animation

- “Core animation is an animation and graphics compositing framework made for speed and efficiency”
- Despite it having the word animation, Core Animation is actually in charge of all rendering on screen. Not just animations.
- Core Animation uses CALayer objects as its main unit of work. UIView objects are just thin wrappers for CALayer’s.
- Layers can be arranged in a hierarchy just like UIView’s, in fact you can build your entire interface using just layers if you wanted.

Core Animation



Built-in Animation Techniques

- Apple provides a 3 different ways to create animations:
  - Adding CAAnimation and its subclasses to a layer (Advanced)
  - Simple block/closure based system using class methods on UIView (Fairly straight-forward)
  - Key-frame animations, which is just a wrapper built around the lower level CAAAnimation method. Uses block/closure syntax as well.

Block/Closure Based

- There are a couple different class methods for the block based animation system.
- One does not have a completion block, all the others do.
- Two of them provide a parameter for a delay before the animation fires.
- One provides an options parameters where you specify the curve type to use (ease in, ease out, etc)
- One provides everything above, plus spring dampening!

Spring animations

- “The type of a motion being used to generate a nice, springy-feeling animation is typically modeled after a damped harmonic oscillation.”
- The key properties of a spring based of a spring’s motion are:
  - Mass: the weight or heft of the object
  - Stiffness: how difficult it is to stretch out the spring
  - Damping: the restrictive force or friction pushing against the object

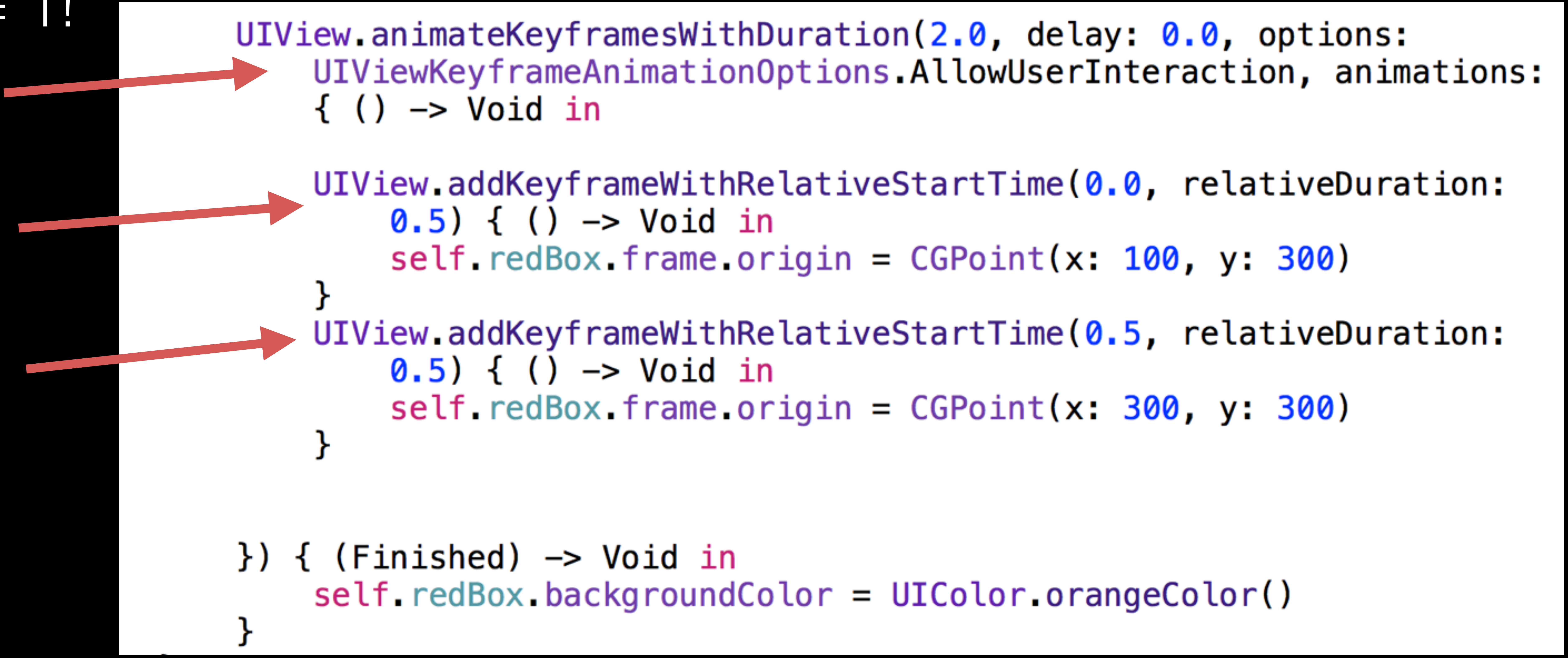
Demo

Key Frame Animations

- Key frame animations allow to you make animations that start and stop at different times, all relative to one duration interval.
- To start a set a keyframe animation, you first create a parent block that all the key frame animations will be created in. This parent block is given a duration, for which all the key frame animations will operate by.
- Each key frame animation is setup with a relative start and stop time inside the parent block.
- The parent block can also be given a set of options for specific animation scenarios.

Key Frame Example

- The duration of the parent is 2 seconds here. The relative duration of the child animations is 0.5. So they will each take 1 second to run since 2 x 0.5 = 1!



```
UIView.animateKeyframesWithDuration(2.0, delay: 0.0, options:
    UIViewKeyframeAnimationOptions.AllowUserInteraction, animations:
    { () -> Void in

    UIView.addKeyframeWithRelativeStartTime(0.0, relativeDuration:
        0.5) { () -> Void in
        self.redBox.frame.origin = CGPoint(x: 100, y: 300)
    }
    UIView.addKeyframeWithRelativeStartTime(0.5, relativeDuration:
        0.5) { () -> Void in
        self.redBox.frame.origin = CGPoint(x: 300, y: 300)
    }


}) { (Finished) -> Void in
    self.redBox.backgroundColor = UIColor.orangeColor()
}
```

Demo

Custom View Controller Transitions

Custom View Controller Transitions

- Debuted with iOS7
- Gives you complete control over the animation while you transition from one VC to another.
- You can even create interactive transitions that are driven by gestures.

Workflow

1. Create an animation controller - This can be any class that conforms to UIViewControllerAnimatedTransitioning protocol. This class will perform the actual animation.
2. Set the transitioning delegate - When you are about to present a view controller, you need to set its transitioning delegate. This is usually just the presenting view controller.
3. Return the animation controller when the transitioning delegate is asked for it.

Animation Controller

- Your animation controller can be any class that conforms to UIViewControllerAnimatedTransitioning protocol
- This protocol has 2 required methods:
  - transitionDuration(transitionContext : UIViewSearchControllerContextTransitioning)
  - animateTransition(transitionContext : UIViewControllerContextTransitioning)

`transitionDuration()`

- A simple method that will return how long the transition will take.
- This should always match up with the duration of the animations you create in the next method.

animateTransition()

- This method is where you actually implement the animations.
- It has one parameter passed in, the transitionContext.
- The transitionContext gives you access to the both the presenting view controller (fromViewController), and the presented view controller (toViewController:

```
 //getting both the view controller we are presenting from (fromVC) and the one we are
presenting (toVC)
 let fromVC = transitionContext.viewControllerForKey(UITransitionContextFromViewControllerKey)
 let toVC = transitionContext.viewControllerForKey(UITransitionContextToViewControllerKey)
```

animateTransition() cont.

- The transitionContext also provides you with something called the containerView
- The containerView essentially acts as the super view for both the presenting view controller's view and the presented view controllers view.
- UIKit automatically adds the view of the presenting view controller, think of that as your starting state of the transition.
- You then add the view of the toVC, and then animate it moving onscreen in some cool way
- Once the animation is complete, be sure to call completeTransition() on the transitionContext.

TransitionDelegate

- The transitionDelegate is mostly just responsible for providing the animation controller at the appropriate time.
- It does this by implementing this method:

```
func animationControllerForPresentedController(presented: UIViewController, presentingController
 presenting: UIViewController, sourceController source: UIViewController) ->
 UIViewControllerAnimatedTransitioning? {
    return self.animationController
}
```

- If you have multiple transitions/segues wired up to this view controller, you would need to inspect the presented view controller to see which one is being presented on screen, and then return the appropriate animation controller

With Navigation Controller

- You can easily create custom transitions for view controllers inside of a navigation stack as well.
- In this case, the navigation controller's delegate will provide the animation controller.
- UINavigationControllerDelegate has a method you will need to implement in order to give it to the animation controller at the appropriate time:

```
//MARK: UINavigationControllerDelegate

func animationControllerForPresentedController(presented: UIViewController, presentingController
    presenting: UIViewController, sourceController source: UIViewController) ->
    UIViewControllerAnimatedTransitioning? {
    return self.animationController
}
```

Demo

Web Tools Wednesday
Express and Routing

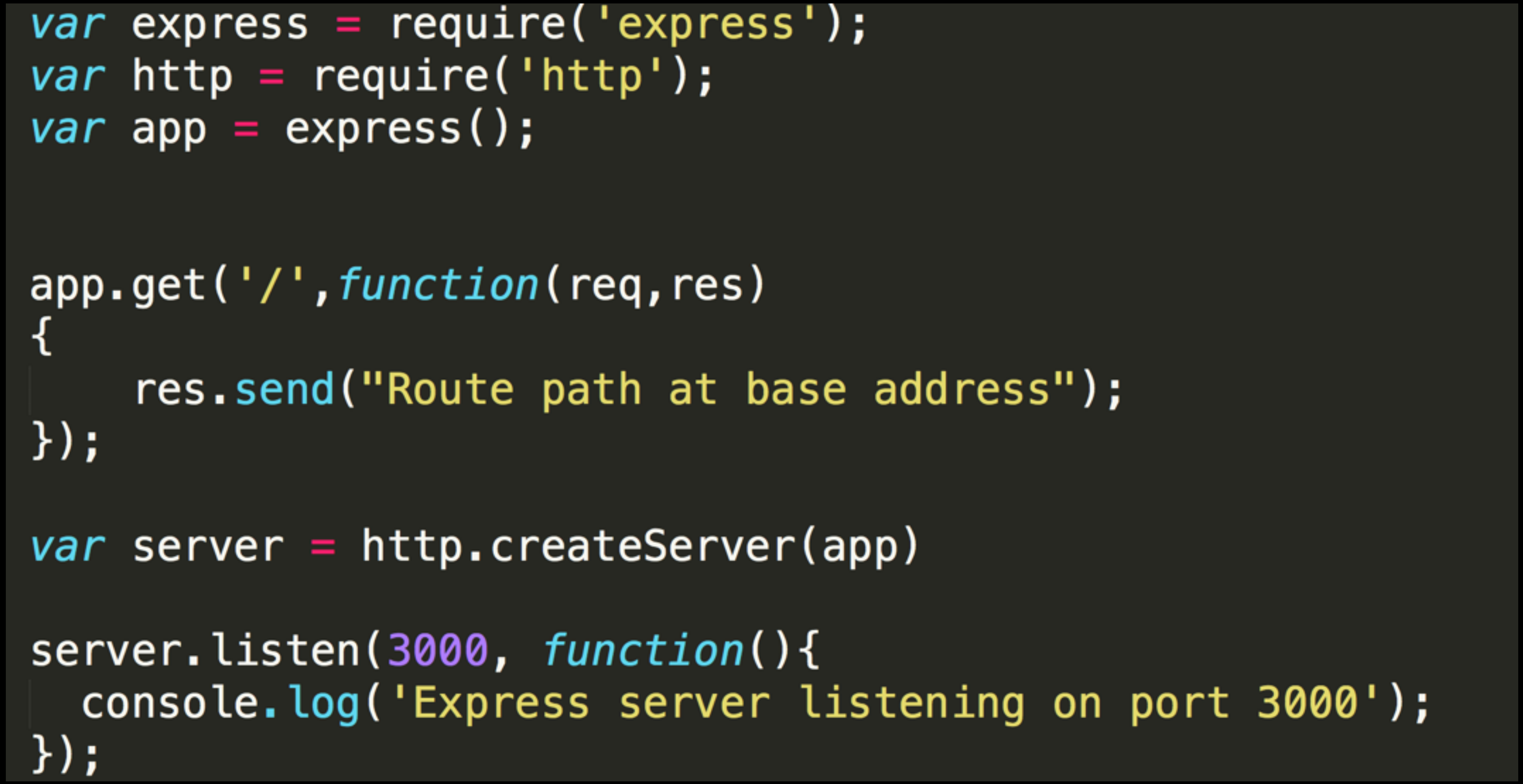
Sourced from https://blog.safaribooksonline.com/2014/03/10/express-js-middleware-demystified/ , http://modernweb.com/2014/04/07/the-basics-of-express-routes/

Express

- “Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile apps”
- It is most popular for its ‘routing & middleware’
- Lets look at routing today

Express Routes

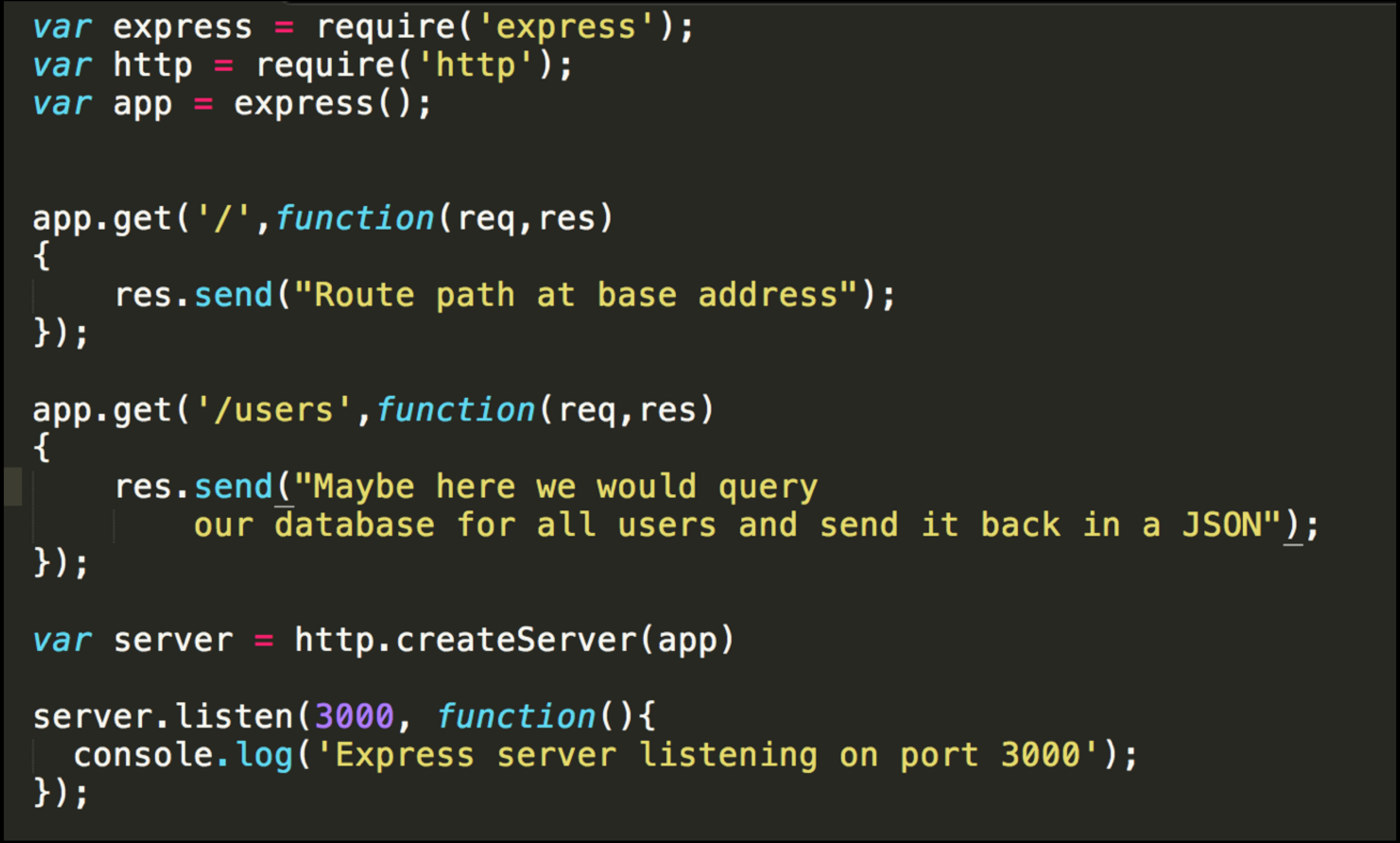
- Express uses HTTP verbs (hey we learned about those!) to perform routing.
- The method names reflect their corresponding HTTP verb.
- Heres a basic script with an express app setup and a single GET route setup on the base endpoint /



```
var express = require('express');
var http = require('http');
var app = express();


app.get('/',function(req,res)
{
    res.send("Route path at base address");
});

var server = http.createServer(app)

server.listen(3000, function(){
  console.log('Express server listening on port 3000');
});
```

Express Routes cont.

- Heres the same app, but with a new /users endpoint added:



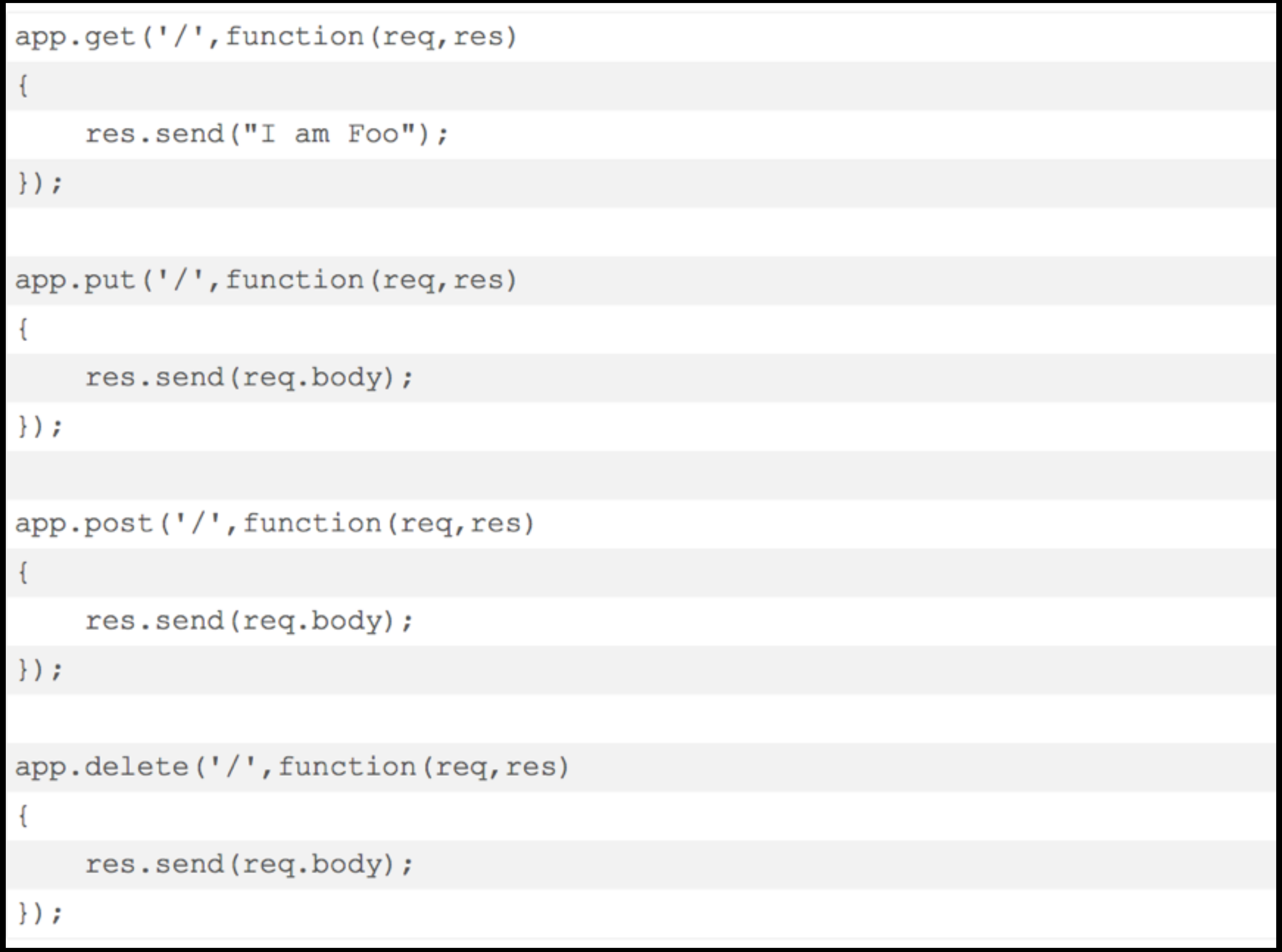
```
var express = require('express');
var http = require('http');
var app = express();


app.get('/',function(req,res)
{
    res.send("Route path at base address");
});

app.get('/users',function(req,res)
{
    res.send("Maybe here we would query
        our database for all users and send it back in a JSON");
});

var server = http.createServer(app)

server.listen(3000, function(){
  console.log('Express server listening on port 3000');
});
```

Express Routes cont.

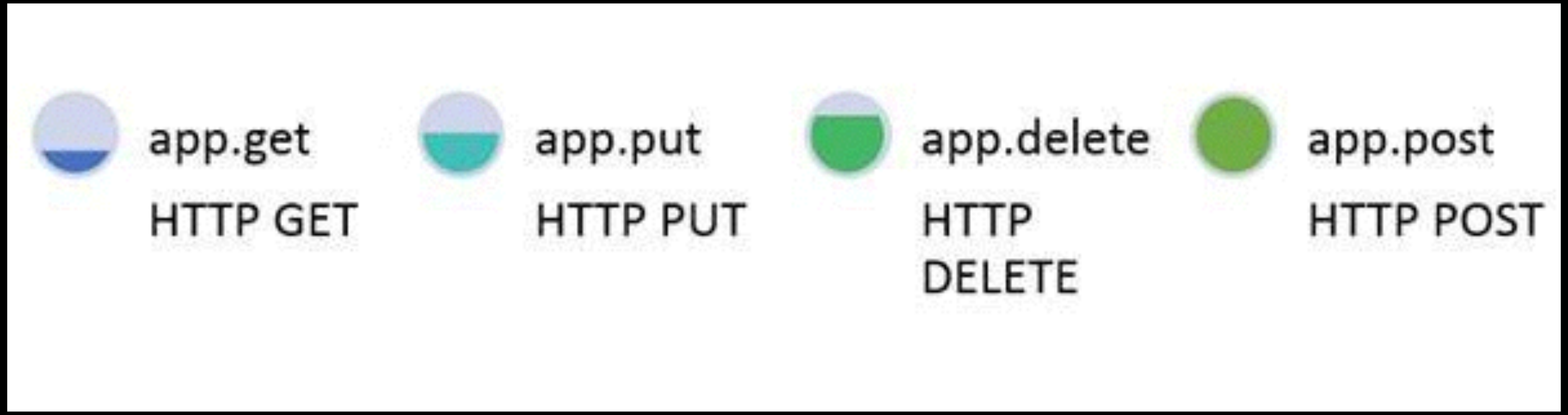
- Use a colon to denote a parameter in your route

```
var express = require('express');
var http = require('http');
var app = express();


app.get('/',function(req,res)
{
    res.send("Route path at base address");
});

app.get('/users',function(req,res)
{
    res.send("Maybe here we would query our database for all users and send it back in a JSON");
});

app.get('/user/:id',function(req,res)
{
    res.send("Here we could pass back the user info JSON for user with id " + req.params.id);
});

var server = http.createServer(app)

server.listen(3000, function(){
  console.log('Express server listening on port 3000');
});
```

Express Other Route Types



```
app.get('/',function(req,res)
{
    res.send("I am Foo");
});

app.put('/',function(req,res)
{
    res.send(req.body);
});

app.post('/',function(req,res)
{
    res.send(req.body);
});

app.delete('/',function(req,res)
{
    res.send(req.body);
});
```



app.get
HTTP GET

app.put
HTTP PUT

app.delete
HTTP DELETE

app.post
HTTP POST

App Idea Presentations