View Frame and Bounds
Core Graphics Fundamental Structures

- **CGPoint**: a structure that contains a point in a two-dimensional coordinate system.
  
  Ex. let pt = CGPointMake(x:3, y:-5)

- **CGSize**: a structure that contains width and height values.
  
  Ex. let mySize = CGSize(width:10, height:5)

- **CGRect**: a structure that contains the location and dimensions of a rectangle.
  
  Ex. let rect = CGRect(x: 3, y: 5, width: 10, height: 5)
  
  or let rect = CGRect(origin:pt, size:mySize)
Frame and Bounds

- *Frame* and *Bounds* are fundamental concepts for all of the elements in the UI.

- Each view has both a frame and a bounds structure. The structure is a CGRect and consists of 4 floats.
  
  - The **frame** of an UIView is the rectangle, expressed as a location \((x,y)\) and size \((width, height)\) relative to the superview it is contained within.
  
  - The **bounds** of an UIView is the rectangle, expressed as a location \((x,y)\) and size \((width, height)\) relative to its own coordinate system \((0,0)\).
Frame
  origin = (0,0)
  width = 219
  height = 300

Bounds
  origin = (0,0)
  width = 219
  height = 300
Frame and Bounds

Frame
origin = (71,50)
width = 219
height = 300

Bounds
origin = (0,0)
width = 219
height = 300
Scroll Views
Scroll Views

- Scroll Views provide a way to present content larger than a single screen.
  - Critical for phones since they have limited screen real estate
  - Also helpful for iPads

- Scroll Views provide a way for moving within the content to view various parts of it.

To implement scrolling:

- Create a `UIScrollView` and define its properties
- Make the `UIScrollView` a subview of the VC’s view
- Make the view you want scrollable a subview of the `UIScrollView`.
Core Location
Core Location

Core Location is a framework that provides the following services for a device to determine:

- its geographic location
- Its altitude
- Its orientation
- Its position relative to a nearby beacon.

The framework uses all available onboard hardware to do this.

- Wi-fi and/or cellular
- GPS
- Bluetooth
- Magnetometer
- Barometer
To begin using Core Location services, you must first import the framework, and then create a \texttt{CLLocationManager} object to start, stop, and manage the delivery of location-related events to your app.

```swift
import CoreLocation

var locationManager = CLLocationManager()
```

Then you must ask the user permission to use Location Services.

```swift
locationManager?.requestWhenInUseAuthorization()
```
This will cause the system to bring up an alert requesting access from the user.

Once the user responds, your app records the user’s response, and will not display this alert again.
MapKit is an API that makes it easy to display maps, plot locations, draw routes and other shapes on top of the map.

- Elements drawn on top of the map are called *overlays*.

You can choose what kind of map is rendered:

- Standard
- Satellite
- Hybrid

You can define pins (annotations) for locations of interest.

- You can define the color of pins.
- You can define *callouts* that show some basic identifying information when the pin is touched.
In order to display a map, you must:

- Import `MapKit`
- Include an `MKMapView` somewhere in a view controller

There are methods available for a number of things, including:

- Adding overlays
- Setting center of the map and the currently visible region of the map
- Changing the map type
- Using coordinates (latitude, longitude) for pin location and/or for a travel path
- Adding annotations
Animation
What does it mean to *animate* something?

- To bring to life
- To cause to appear as if it’s moving or changing

In the context of iOS applications, this means to modify aspects of the user interface in a special way as to produce the appearance of action.

Why would we want to animate something?

- It draws the user towards things that change
- It indicates importance at a particular moment
- It makes your app look cool, fun, or polished – which can be a differentiator
You can animate the following properties of a UIView derived object:

- **frame** – move or scale the view (relative to its superview)
- **bounds** – move the view’s contents within the view
- **center** – move the view relative to the screen
- **transform** - scale, rotate, or translate the view relative to its center point
- **alpha** - gradually change the transparency of the view
- **backgroundColor** - change the view’s background color
- **contentStretch** - change the way the view’s contents are stretched to fill the available space
The basic UIView animation method is `UIView.animate`: 

```swift
UIView.animate(
    withDuration: <duration>,
    delay: <delay>,
    options: <options>,
    animations: {
        <animation code>
    }
    completion: {
        <completion code>
    }
)
```
UIView.animate (cont.)

duration: how long in seconds to run the animation
delay: how long to wait until starting the animation
options:
  .curveEaseInOut  begin slow, accelerate, end slow
  .curveEaseIn     begin slow, accelerate to end
  .curveEaseOut    begin quickly, slow to end
  .curveLinear     even over the duration
  .repeat          make the animation loop forever
  .autoreverse     animate forward, then reverse

animation code:
 identifies the ending value for the selected attribute(s)

completion code:
 code to be executed at the end of the animation
Alpha Animation:  Fade Out

Adjust the *alpha*:

```
// Starting alpha value
self.labelName.alpha = 1.0

UIView.animate(
    withDuration: 3.0,
    animations: {
        self.labelName.alpha = 0.0
    }
)
```
Adjust the `center`:

```swift
// Starting center value
self.labelName.center.x = self.view.center.x

UIView.animate(
    withDuration: 3.0,
    animations: {
        self.labelName.center.x +=
            self.view.bounds.width
    }
)
```
Center Animation: Spinning

Adjust the `transform`:

```swift
UIView.animate(
    withDuration: 3.0,
    animations: {
        // 180 degree rotation
        self.labelName.transform = self.labelName.transform.rotated(by: CGFloat(Double.pi))
    }
)
```