CS121 Tutorial 2
Intro to views and re-intro to buttons.

Purple bubbles give you information you’ll need to know.

Yellow Bubbles tell you what to do.

Orange bubbles tell you what you’re not expected to understand yet. 😊

Green bubbles are questions for you to look up and answer!

8/31/13
Create a new project but this time make it an Empty Application.

Name your project ViewTest.
1. Open the Supporting Files folder.

2. Open main.m.

This is our main! Main launches our app by instantiating the AppDelegate class.
An empty application still has an AppDelegate, which launches the app, but no ViewController is generated.

Every app needs a window. The window owns and manages the views that an app displays to the screen of the device. Here is our window.

Open the AppDelegate header file.
Run the app and check out the simulator.

The white region is our window.

Even an empty app has a status bar.
1. Open the AppDelegate source file.

This line instantiates the window.

This line makes the window white.

2. Change whiteColor to redColor.
Run the app and check out the simulator.

Now our window is red.
Select the File tab then New and New File.
1. Select Cocoa Touch.
2. Select Objective-C Class.
3. Click Next.
2. Make it a subclass of NSObject.

3. Click Next.

1. Call the class SimpleView.

NSObject is the root Object class of Objective-C.
Open the SimpleView header file.

Huh?
Why didn’t we just tell Xcode that to begin with?
Because Xcode would have produced a lot of template code we don’t want right now!

2. Change SimpleView so it inherits from UIView rather than NSObject.

Google: What template code would Xcode have produced?
Now SimpleView inherits from UIView. Next we’ll instantiate our class.
1. Open the AppDelegate source file.

2. Import our class header.

3. Insert this code.

```
#import "AppDelegate.h"
#import "SimpleView.h"

@implementation AppDelegate

@synthesize window = _window;

-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window = [(UIWindow alloc) initWithFrame:[UIScreen mainScreen] bounds];
    // Override point for customization after application launch.

    CGRect frame = CGRectMake(50, 50, 100, 100);
    SimpleView view = [[SimpleView alloc] initWithFrame: frame];
    [view setbackgroundColor: [UIColor whiteColor]];
    [self window addSubview:view];

    self.window.backgroundColor = [UIColor redColor];
    [self.window makeKeyAndVisible];
    return YES;
}

@end
```

Document this code
Run the app.

This is our new view. Everything displayed is in a view.
A view is like a subwindow. The key steps to creating a view are to:
1. Define its “frame” – the region it will occupy.
2. Add it as a subview.

It was not critical to set its background color but otherwise it would have been the same color as the window and we wouldn’t have been able to see it – This is a key error made by many.
We can have multiple views!

Insert this code to instantiate a new SimpleView object
Run the app.

This is our new view. Views are drawn in the order they are added as subviews, so the blue view is drawn on top of the white one which is on top of the red one. (There is a way to reorder the views.)
A view can have subviews.

Make view2 a subview of view1.

Don’t make the same spelling error.
Run the app.

Is this what you expected?
A view’s frame is defined relative to its parent’s frame.

On the left, the blue view is defined relative to the window’s origin, which is the top left of the screen. (The origin is actually occluded by the status bar.)

On the right, the blue view is defined relative to the red view’s origin, which is at 50,50 in the window’s frame.
Here are” UML Object diagrams” of the two approaches!
If you ran the app with the console open, you’d notice that we are being warned our app should have a root view controller.
What we did was an experiment. Until you become an iOS expert, you would be wise to use the template code. In some cases you may want to use template code forever.
Create a new single view iPhone app named tutorial2.

A ViewController class is automatically defined for us.
Open the AppDelegate source file.

We still have a window but this time the template code also instantiates our ViewController class.

The ViewController object will be the rootViewController for the app and the app’s windows.
Run the app.

The UIViewController class has a pointer to a view. When the class is instantiated this “main view” is created as well. The default background color of the main view is gray.
Open the ViewController source file and modify the viewDidLoad method so the main view is white.
Run the app.

Now the rootViewController’s main view is white.
Now we are going to give our view controller a subview that will eventually hold our Sudoku grid.
Create a new class Grid that inherits from UIView.
A UIButton subclass should implement this method. It doesn’t have to do anything other than call its parent’s method of the same name.

Add this line to set the background color to black.
1. Open the ViewController header file.

2. Import Grid.h.

3. Create a member variable to store our view.

We can use a UIView* because Grid.h doesn’t significantly modify UIView. If we want to give Grid additional properties, we’d change this to a Grid*.
Open the ViewController source file.

Add these lines to create theGrid and add it as a subview.
In creating the frame for our grid, we size it relative to the main view. This allows our implementation to work for iPhone and iPad, whether in portrait or landscape mode.
Run the app.

This is our grid view. You’ll need to refine the location and size once you develop and test your UI design.
Open the Grid header file.

Let’s add a button to our grid.
Open the Grid source file.

UIButton is a subclass of UIView. So we need to give it a frame. We can also set its background color.

Add this code to create the button and add it as a subview.
Run the app.

This is our button. It doesn’t do anything yet. We have to configure it appropriately.
Open the Grid header file.

Declare a buttonPressed method.
1. Open the Grid source file.

2. Set the “target” for a button press.

3. Define the buttonPressed method.

The NSLog command writes an NSString to the console.

We’ll talk about “targets” later.
Run the app.

Press the button
A message appears in the console window.
Summarize what you did