CS290.02
Homework 2 - Recipe App v1 with SwiftUI

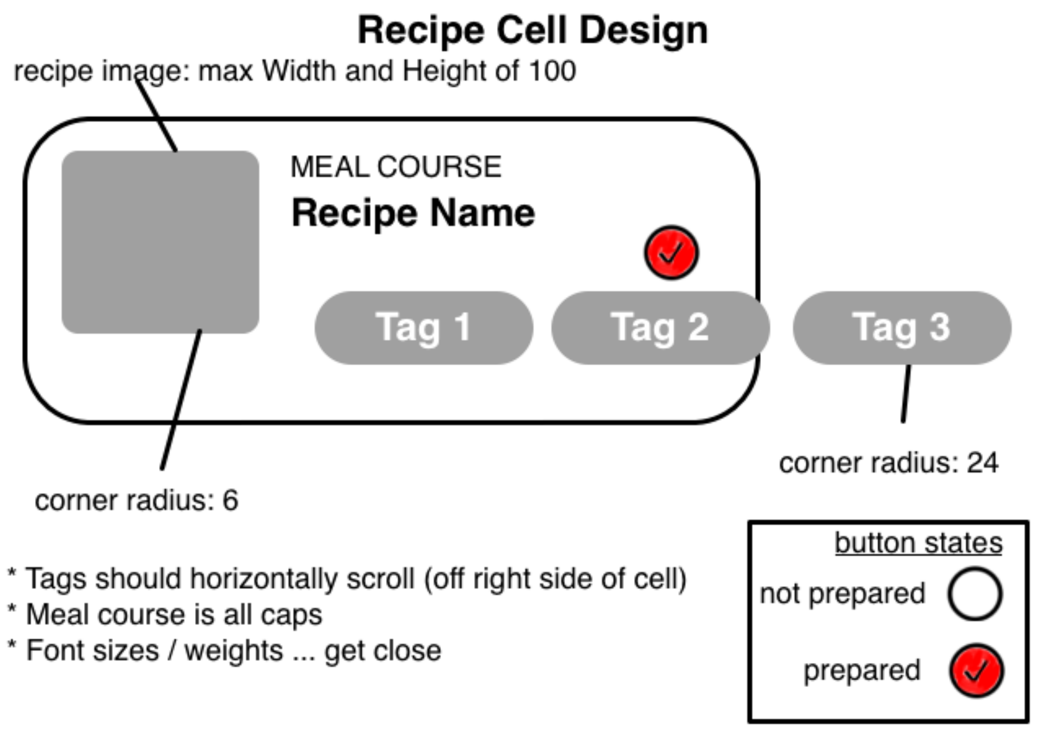
The goal of this homework is to get a recipe app up and running. Your first app will display a list of recipes, and you can tap on a recipe to navigate to a detail page. Your challenges:

- Learning SwiftUI layout
- Passing a Binding to the detail view so you can change the date the recipe was last prepared by tapping a Button.
- Breaking up large views you create into sensible components.

You will be adding files from both the first Homework and the RecipePreviewData file included with this homework. You could create empty Swift files in Xcode and then copy and paste the text in. Or you could drag the files from your Finder into Xcode. If you do the latter, a dialog box will appear. Make sure the "Copy items if needed" checkbox is checked and that the "Add to targets" box has your iOS target checked and click "Finish".

1. Include your data models from Exercise 1 (Recipe, Ingredient, etc) in your app in a folder called Models. Do not include the previous previewData; we have provided an updated version with some small position integer corrections. Same deliciousness though.
2. Add a new property to your Recipe model: lastPreparedAt: Date? = nil
3. Include the file "RecipePreviewData" from the homework in your Models folder. Note that this file is just using our **extension Recipe** approach as usual. It doesn't matter that it is in a different file.
4. Create a list view leveraging the previewData, where each cell which matches the spec given by the designer (see accompanying illustration below)

   - A Thumbnail on the left side of the cell
   - The title of the recipe
   - The mealCourse of the recipe
   - The tags of the recipe, which will scroll horizontally
   - An indicator whether or not the recipe has been prepared previously


Recipe Cell Design
recipe image: max Width and Height of 100
MEAL COURSE
Recipe Name
Tag 1
Tag 2
Tag 3
corner radius: 24
corner radius: 6
* Tags should horizontally scroll (off right side of cell)
* Meal course is all caps
* Font sizes / weights ... get close
button states
not prepared
prepared

Hint 1: To display the mealCourse, you'll want to have the MealCourse enum representable by a raw value. Change the enum definition to "enum MealCourse: String. You can now get a string by asking the enum case for its rawValue. And if you don't like the rawValue, you can define a preferred string for each case. For example:

```
enum MealCourse: String {
    case appetizer
    case mainDish = "main"
}
```

Hint 2: Tags scrolling? Sounds like we need a ScrollView. I wonder if it has some settings in its parameters that will help us with the scrolling direction.

5. Create a navigationLink for each recipe with a DetailView destination (remember, you'll want to make sure your list is wrapped in a NavigationView). Include a navigation bar title that is "Recipes - " followed by your NetID. i.e., the top of Jon's Screen would say "Recipes - jrp96"
6. Create a detail view for the recipe. The creative lead did not provide a design spec[1], so you're on your own. You'll want to include:

   - Recipe image
   - Recipe title
   - **A Button that changes whether lastPreparedAt has a value or does not**
   - Ingredients, sectioned or unsectioned as appropriate
   - Instructions.

   Some (I hope helpful) notes:
   a) it is fine for the recipe amounts to be expressed as decimals even though that is not typical for a recipe. Displaying the amounts as fractions will require some very mathematical third party code, which we can perhaps revisit at some point
   b) Don't worry about numbering the instructions, although you might expect them to be. There are ways to include an index with ForEach which you could use to number them, but that is not complexity we will tackle now.
   c) Note that in class we've been using buttons and toggles to change Booleans. Here you are using a button to change whether a value has a date or is nil. There's nothing clever to do here … there is no secret automatic way to do it. Use the button action to set the value and don't overthink it. (And this is typical of some code bases: why have a boolean that says if you've ever prepared the dish when you can have a date for lastPreparedAt which captures slightly more information?)
   d) You may want to create a function on your view that creates a string for each recipe component (e.g., "1 cup flour"). (This is not a requirement, just one suggestion.) A function with the signature:

   ```
   func componentDisplay(_ recipeComponent: RecipeComponent) -> String
   ```

   As you iterate through the recipe components in a section, you could call that function to produce the string.

[1] Their band had a gig that night so they got distracted and didn't get around to it. But you're the developer, so you have to stay until it's done and fully functional. It's the life you chose.

7. Include an edit button in a navigation toolbar on the detail view that displays a Modal Sheet. For now, have the modal sheet display the text “Edit Form to Come”. You can just swipe down to dismiss that modal for now.
8. In all of the above efforts, use your judgement to break down large views into smaller components. If you have one long ListView and one long DetailView (or whatever you decide to call them) … you will not get 100% on the homework.
9. Be diligent with your code organization indentation; better than I have been sometimes while live coding in class. Hint: SwiftUI can get you 99% of the way there if you Command-A to select all the text in a file and hit Ctrl-I to get Xcode’s best take on indentation. You are writing code *to be read*. Not just by the TAs and myself, but by a theoretical **you** six months from now. Your future self will have forgotten half the things you did and why you did them, and they will thank you for leaving your code in a readable and maintainable state.

OTHER THINGS TO REMEMBER
Images: If you want to do any sizing of any images, you should start with the resizable() modifier.