



Description

In this assignment you will construct SwiftUI Views for a simple application that tests the user's multiplication skills. It requires a little bit of coding, but the primary goal of the assignment is to familiarize you with the tools we'll be using in the class.

Your app will eventually have the following functionality: Upon pressing a button labeled "Start" a multiplication problem is presented by giving a multiplicand and a multiplier (each of which is a random number between 10 and 20, inclusive). The answer field should be blank, but four buttons below the problem display should display 4 possible answers, one of which is correct. The other three should be within 5 of the correct answer with no duplicates. Upon the user selecting an answer, the app should display the correct answer, indicate whether the user's answer was correct, and update a running total of correct answers. Upon pressing the button (now labeled "Next"), another problem should be presented and labels appropriately updated. The game continues until 5 problems have been answered, at which point the button should be labeled "Reset". Pressing the button at this point resets the app to its initial state.

Remember: for this assignment, you only need to create SwiftUI views. You will add the functional in Assignment 2.

Walk Through

1. In Xcode, create a new Xcode project, choosing the iOS platform and "App" template. Give your app a meaningful name because this will appear under the app icon on the springboard. "Assignment 1" is not a good name. Be sure to choose *SwiftUI Interface*, *SwiftUI App Life Cycle* and *Swift Language* from the drop-down lists. Leave the *Use Core Data* box unselected but select the *Include Tests* box. Click Next. Be sure to place the project in the folder under source control and linked to your Github repository. All your assignments will go in this folder. The option to add source control should be grayed out because Xcode will recognize that this folder is already under source control.
2. Open ContentView.swift and make sure you have both the Editor and Canvas open. If only the Editor is visible use the keyboard shortcut `⌘⇧C` to display the Canvas. If a preview is not visible, click 'Resume Previews' in the top right corner of the canvas (or use the keyboard shortcut `⌘⇧P`). You should see a preview of a blank view that displays "Hello, World!"
3. Start adding views, stacks, and modifiers containing placeholder data to ContentView and extracting subviews when necessary. Use the preview in the canvas to make adjustments while designing. See the above illustration for guidance, but be creative with your design! Your interface should display the following:
 1. The multiplicand, multiplier, multiplication symbol and horizontal line
 - A Rectangle or RoundedRectangle view with a frame modifier can be used for the horizontal line
 2. An indication of the user's progress and score
 - Users should be able to see what problem they are on
 - Users should be able to see how many and which questions they have answered correctly so far
 3. Buttons which display the answer choices for the displayed question



4. A button which can be clicked to go to the next question (after the current question has been answered)
 - This button will be disabled or hidden when it cannot be used
4. You should not implement all of this in one big view — you should modularize where it makes sense. Ideally, each view should be responsible for one part of the interface.
 - You can command-click a view (in the editor or canvas) and select Extract Subview to do this quickly if that option is available.
 - Otherwise, declare a new struct which conforms to View manually:

```
struct NewView: View {  
    var body: some View {  
        Text("created a new view")  
    }  
}
```

5. Once you've broken up the interface into single-responsibility views, add properties to each view that will determine what the view will display. Think about what data each view will need 'to do its job' and create a property for each piece of required data.
 - Adding properties will change your view's initializer and you will have to adjust any existing declarations of your view in your code. For now, pass in placeholder data so you can see what your interface will look like in the canvas preview.
6. Add an app icon to your project. Select the Assets.xcassets folder from the File Navigator and then select the AppIcon from the sidebar in the editing window. A template for adding icons will appear. All your submissions should include appropriate icons. No need to get fancy here. With <https://appicon.co/>, you can generate an AppIcon set from an uploaded image. Generate an icon set for iPhone and iPad and drag it into the Assets.xcassets editor and make sure the icon set with your images is named "AppIcon".

Testing


Usually, testing of the application is imperative — this assignment won't require much testing. Just make sure your application builds and runs on the simulator. Be sure that all the required features have been implemented.

1. Your project should build without errors or warnings.
2. Test your layout on multiple iPhone sizes on the simulator. You need only worry about portrait mode, not landscape. You do not need to test it on iPads.
3. The app should have a clean, consistent and intuitive user interface. Fonts and colors should be appropriately chosen.
4. Each of the bullet points in the Walk Through section above will be considered to verify that you've completed the assignment correctly.



Hints

Most assignments will have hints sprinkled throughout the assignment or in a separate section like this.

1. You will use String Interpolation to display numeric data in Text views.
2. You can use the `.frame` modifier to specify the width and/or height of a view. But use this as little as possible.
3. You should leave the action closure of your buttons empty for this assignment.
4. For this assignment, all of your views should be *stateless* (this won't be true for the full app!). You can make a view stateless by making sure all its properties are *let* (immutable) properties. This means that once the view is created, it cannot change – a new view will be rebuilt if the data passed into the initializer changes.
5. Customize your app's appearance! The default button and fonts are quite plain. A background image or color might also contribute to a custom look. Use the `.background` and `.foregroundColor` modifiers.
6. Use special symbols in strings (like a  or **✕**). Use `^⌘Space` to bring up the special character viewer.
7. Spend some time poking around Xcode and SwiftUI views and modifiers.
8. Occasionally Xcode might get confused or in a funky state. Credentials appear invalid, Bots do not show up, etc. Quitting and restarting Xcode is often a fix for these cases.

Troubleshooting

Assignments may also have a section for troubleshooting to call out common pitfalls or areas to watch out for.

Submission

Prepare your project for submission:

1. Remove all breakpoints, print statements and any other cruft.
2. Make sure your project compiles cleanly from scratch: choose **Clean** from the Product menu in Xcode, then Build and Run your program. Be sure to remove all warnings.
3. Check that all resources were copied into your project folder.
4. Ensure that you have merged your work back onto the master branch. **Commit and push your project**
5. Create a Bot and run an integration to verify that your code on GitHub is complete.