

Data Types & Functions Build in SwiftUI

Welcome to your first independent project in SwiftUI! In this project, you will practice using data types and functions to build a simple SwiftUI app. Don't worry if this seems overwhelming at first – take it step-by-step and remember to have fun!

Project Goal

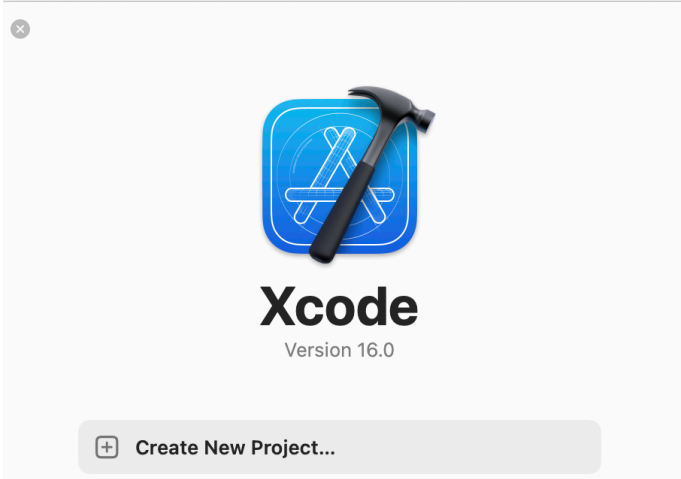
By the end of this project, you'll:

- Create a SwiftUI app with basic interaction using data types (such as integers and strings).
- Use functions to perform tasks like calculations and showing messages.

You'll be building a simple "Calculator App" that takes two numbers as input, performs a calculation, and displays the result.

Setting Up Your Project

1. Open Xcode and create a new SwiftUI App project.
2. Name your project SimpleCalculator.
3. Once your project is created, open the ContentView.swift file – this is where you'll build your app's interface and logic.



Choose a template for your new project:

Multiplatform

ios

macOS

watchOS

tvOS


visionOS

DriverKit


Other

Filter


Application




App




Document App




Game




Augmented Reality App




App Playground



Sticker Pack App



iMessage App



Safari Extension App

Product Name:

SimpleCalculator

Team:

Noah Carpenter (Individual)

Organization Identifier:

Noah Carpenter

Bundle Identifier:

Noah-Carpenter.SimpleCalculator

Interface:

SwiftUI

Language:

Swift

Testing System:

None

Storage:

None

☐ Host in CloudKit

Define Your Variables (Data Types)

What:

We define variables to store the numbers for calculation and their result. These variables can change when the user interacts with the app.

Why:

@State makes the variables reactive, so the UI automatically updates whenever their values change.

Code:

```
// Define your variables inside the ContentView struct.  
@State private var number1: Int = 0 // Stores the first number input  
by the user.  
@State private var number2: Int = 0 // Stores the second number input  
by the user.  
@State private var result: Int = 0 // Stores the calculated sum.
```

@State: This SwiftUI property wrapper allows the variables to update the UI when their values change.

number1 and number2: These are integers (Int) where the user will input their numbers.

result: This will hold the sum of number1 and number2.

Create a Function for the Calculation (functions)

What:

The function adds the two numbers (number1 and number2) and updates the result.

Why:

Encapsulates the calculation logic for clarity and reuse.

Updates the result variable, which will automatically reflect in the UI.

Code:

```
// Define a function to calculate the sum of the two numbers.  
func calculateSum() {  
    result = number1 + number2 // Adds the two numbers and assigns the  
sum to `result`.  
}
```

calculateSum: A custom function that performs the addition.

Inside the function:

The values of number1 and number2 are added.
The result of the addition is stored in result.

Build the User Interface

What:

Create an interface where users:

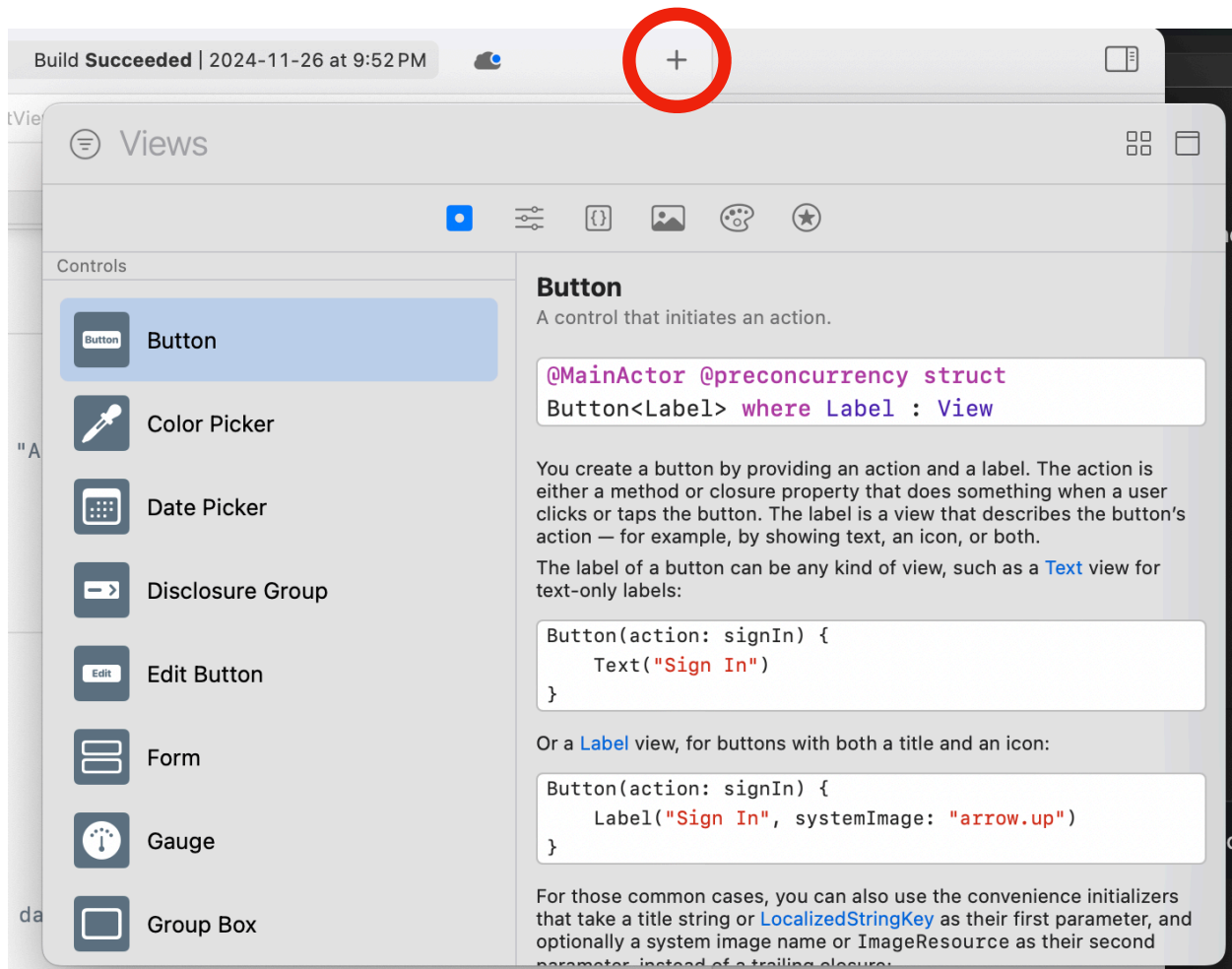
- Enter two numbers.
- Press a button to calculate the sum.
- See the result displayed.

Why:

Provides interactivity for the user.

Organizes elements into a vertical stack (VStack) for clarity.

Code (Reminder use the Code Library to easily add templated UI Elements!!)



```

// Define the body of the ContentView, where the UI is built.
var body: some View {
    VStack { // Arranges UI elements vertically.
        // Input field for the first number.
        TextField("Enter first number", value: $number1,
format: .number)
            .textFieldStyle(.roundedBorder) // Adds a border for
better visual appearance.
            .padding() // Adds spacing around the text field.

        // Input field for the second number.
        TextField("Enter second number", value: $number2,
format: .number)
            .textFieldStyle(.roundedBorder)
            .padding()

        // Button to trigger the calculation.
        Button(action: calculateSum) { // The `action` triggers the
`calculateSum` function.
            Text("Calculate Sum") // Button label.
                .font(.title) // Makes the text larger and more
prominent.
                .padding() // Adds spacing inside the button.
                .background(Color.blue) // Sets the button's
background color.
                .foregroundColor(.white) // Sets the text color to
white for contrast.
                .cornerRadius(10) // Rounds the button's edges.
        }

        // Text to display the result.
        Text("Result: \(result)")
            .font(.title2) // Sets a slightly smaller font size for
the result.
            .padding() // Adds spacing around the text.
    }
    .padding() // Adds overall padding to the VStack.
}

```

Input Fields (TextField):

TextField("Enter first number", value: \$number1, format: .number)

The placeholder text prompts the user to enter a number.

\$number1: Links the input directly to the number1 variable.

textFieldStyle(.roundedBorder): Adds a clean border around the field.

padding(): Adds spacing around the field.

Repeat for number2 with the same structure.

Calculate Button:

Button(action: calculateSum): The button triggers the calculateSum function.

Inside the button:

Text("Calculate Sum"): Describes the button's purpose.

Styling: Adjust font size, add padding, set a background color, and round the corners.

Display Result (Text):

Text("Result: \(result)"): Dynamically shows the result.

Styling: Use a slightly smaller font size and add padding for spacing.

VStack Layout:

Groups the input fields, button, and result text vertically.

Adds padding to the entire layout for a neat appearance.

Summary of Flow

1. User inputs two numbers into the TextFields.
2. Pressing the "Calculate Sum" button runs the calculateSum function.
3. The result updates and is displayed in the Text below the button.

Test Your App

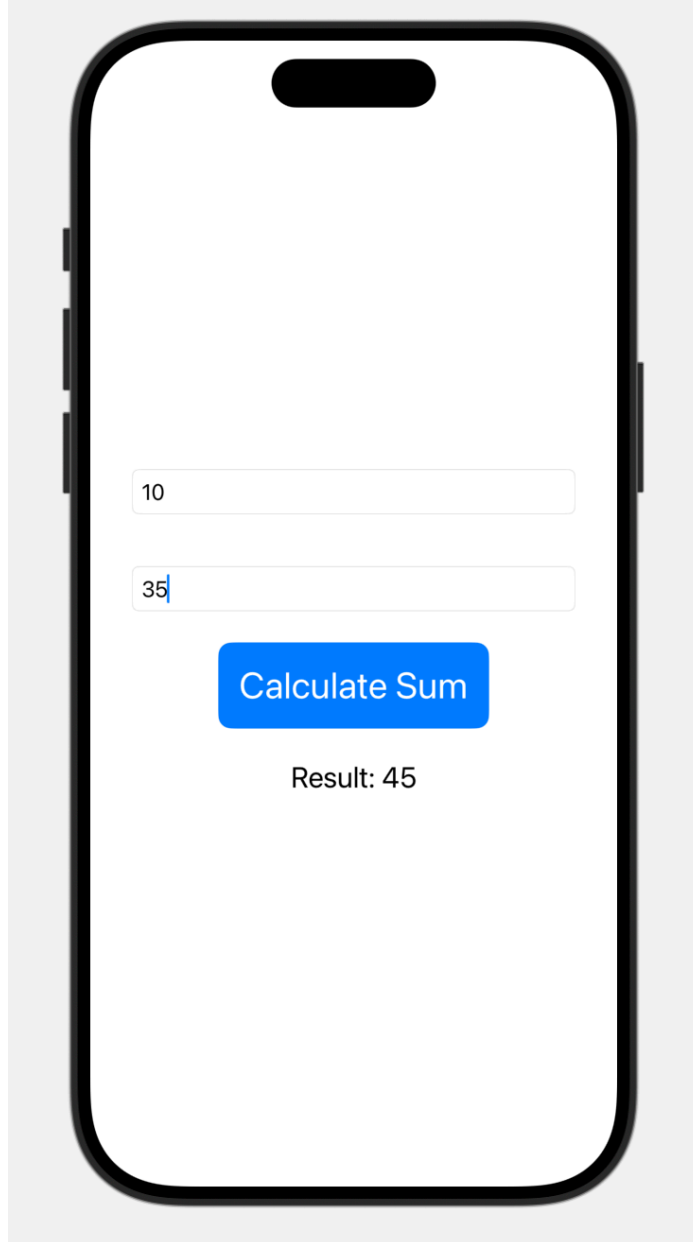
- Run your app using the Simulator (or your connected device).
- Enter two numbers into the text fields.
- Press the "Calculate Sum" button and see the result displayed below.

If you get an unexpected result, check your code for typos, especially in the function or variable names.

Experiment with Other Calculations

Now that your basic calculator is working, experiment with other operations:

- Create new functions to subtract, multiply, or divide the numbers.
- Update the UI to allow the user to choose which operation to perform (e.g., by using buttons for addition, subtraction, etc.).



After adding a new calculation please submit your completed app project

Submitting your work

Step-by-Step Instructions:

1. Locate Your Project Folder

- Open Finder.
- Navigate to where your Xcode project is saved.
- Look for a folder with the name of your project and a blue Xcode icon inside.

2. Right-Click (or Control-Click) the Project Folder

- Right-click (or Control-click) on the folder.
- Select “Compress [ProjectName]” from the dropdown menu.

3. Wait for the Compressed File

- Finder will create a new file next to your folder.
- The new file will have the same name as your project, but with .zip at the end.
- This is your compressed file. It contains everything in your project folder.

4. Submit the File

- Open your class platform or email where you’ll submit the project.
- Attach the .zip file and follow your teacher’s instructions.

General FAQ

- What is @State in SwiftUI?

@State is a property wrapper used in SwiftUI to declare variables that are bound to the view’s state. When the value of a @State variable changes, the view updates automatically.

- Why do we use TextField for input?

TextField is the SwiftUI component that allows users to input text. Here, we use it to enter numbers, but we specify the format to accept only numbers using .number.

- How do I know if my function is working?

Ensure that the function is properly called when the button is pressed. You can add print statements inside the function to verify that it’s running and performing the correct operations.

Common Issues & Solutions

- Issue: The app doesn’t show the correct result.
- Solution: Check if the function is correctly updating the result variable. Ensure that

@State variables are used properly for reactivity.

- Issue: The TextField is not accepting numbers.

• Solution: Make sure you’ve used .number formatting on the TextField to accept numeric input only. This is critical for arithmetic operations to work.

- Issue: The layout is not responsive or looks off on different devices.

• Solution: Ensure that you’re using proper layout modifiers like .padding() and .frame() to adjust the view size. SwiftUI is declarative, so it should automatically adjust for different screen sizes, but ensure you’ve structured your layout properly.

