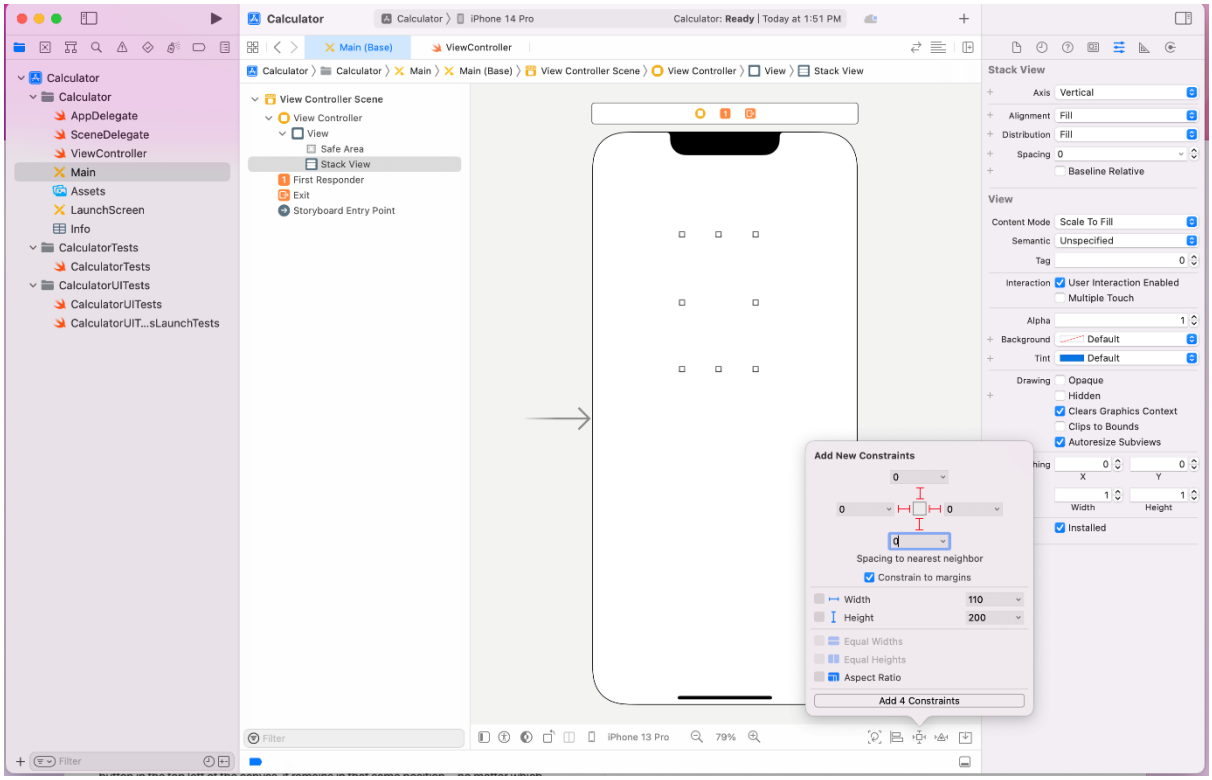
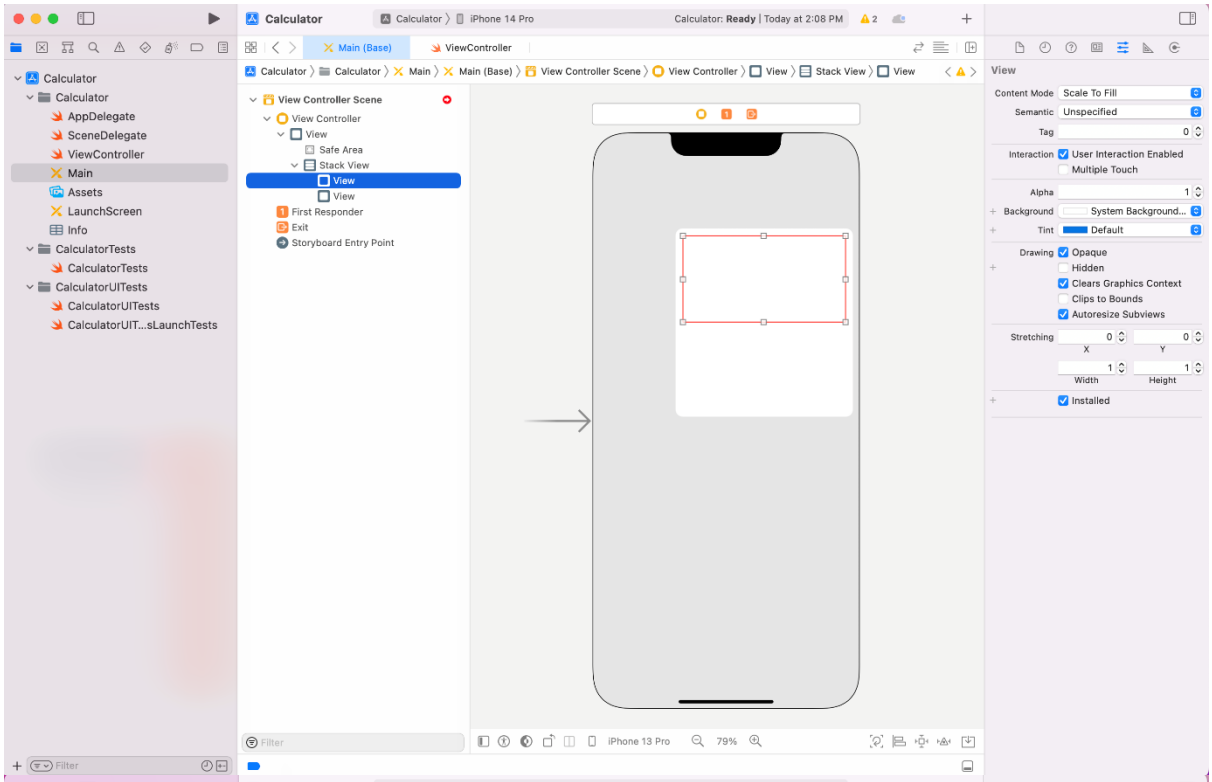
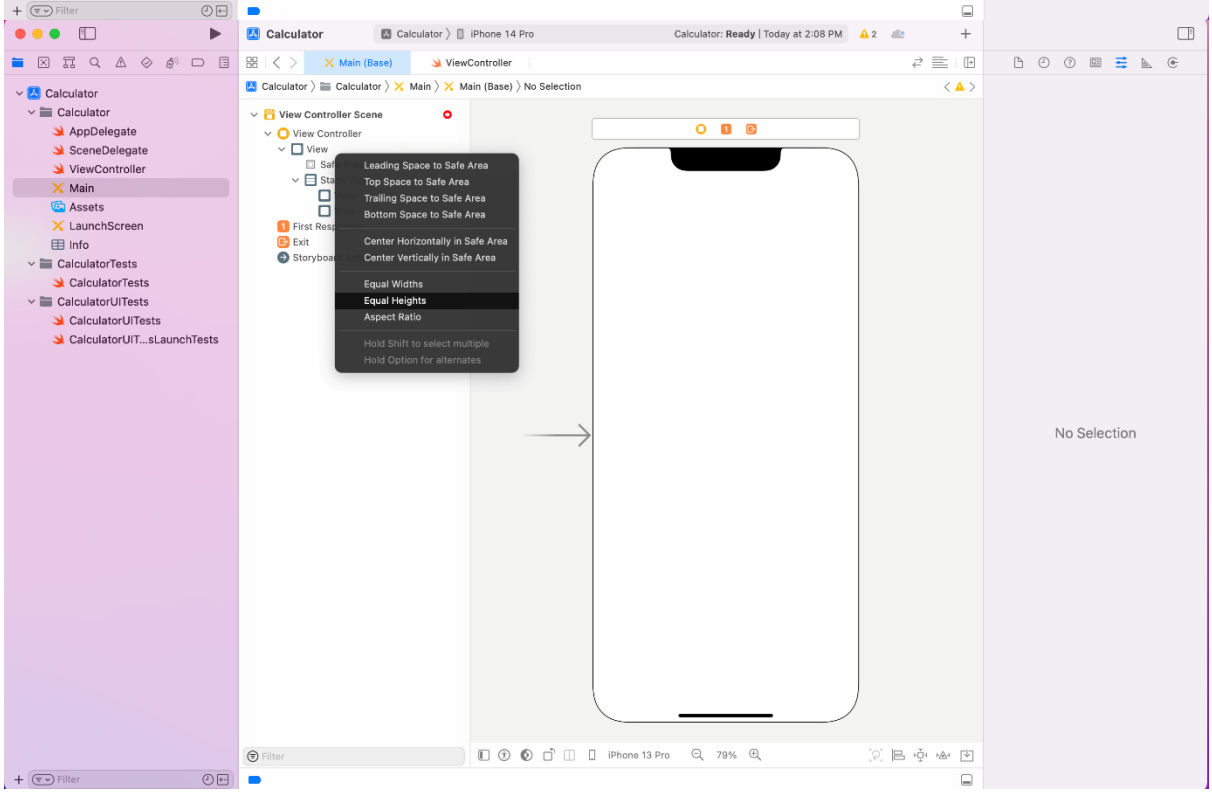
Name: Khushi Nitinkumar Patel
PRN: 2020BTECS00037
Batch: T2

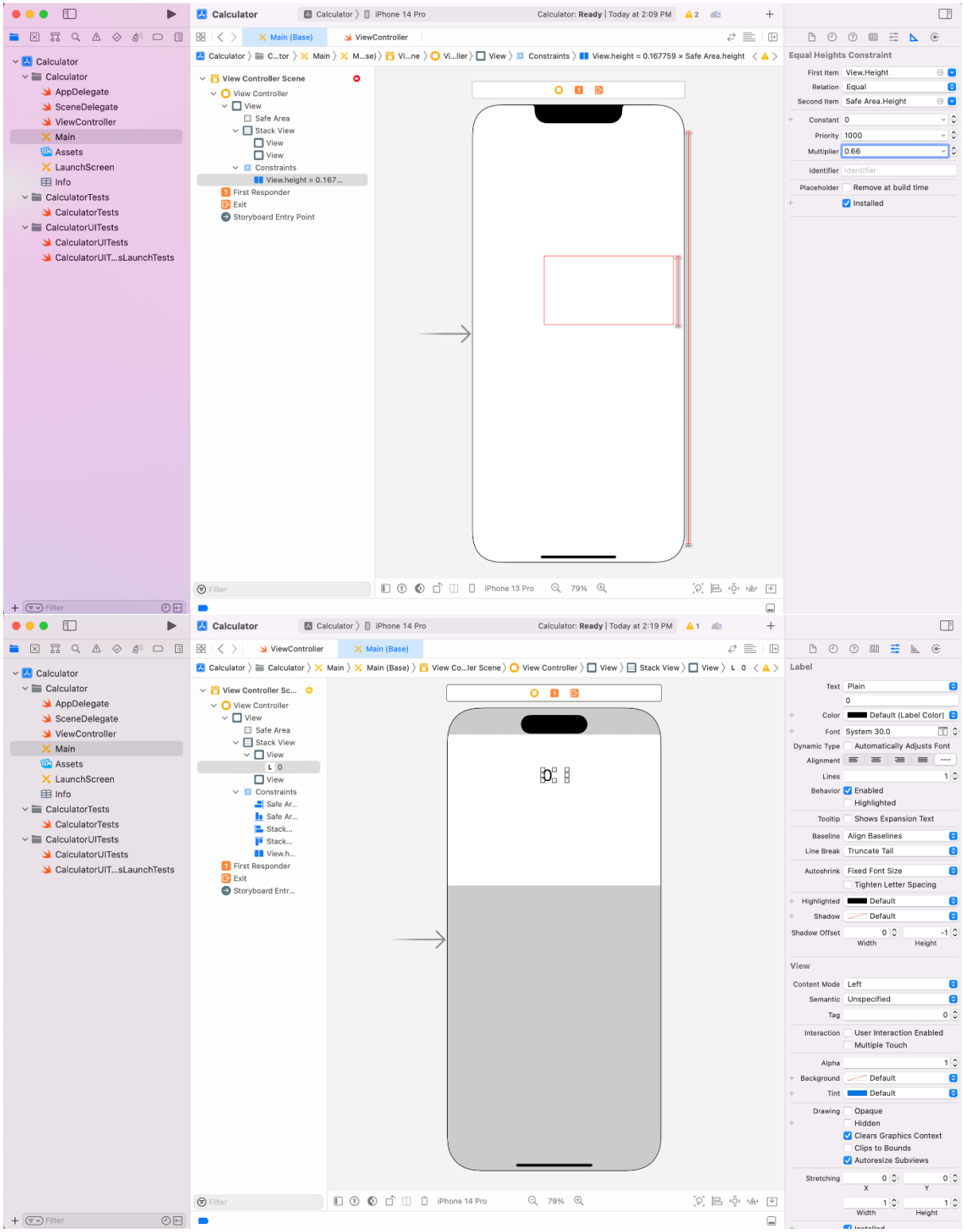
1. **Calculator**

Design a layout in storyboard of Calculator using the auto Layout and Stack View Concepts.

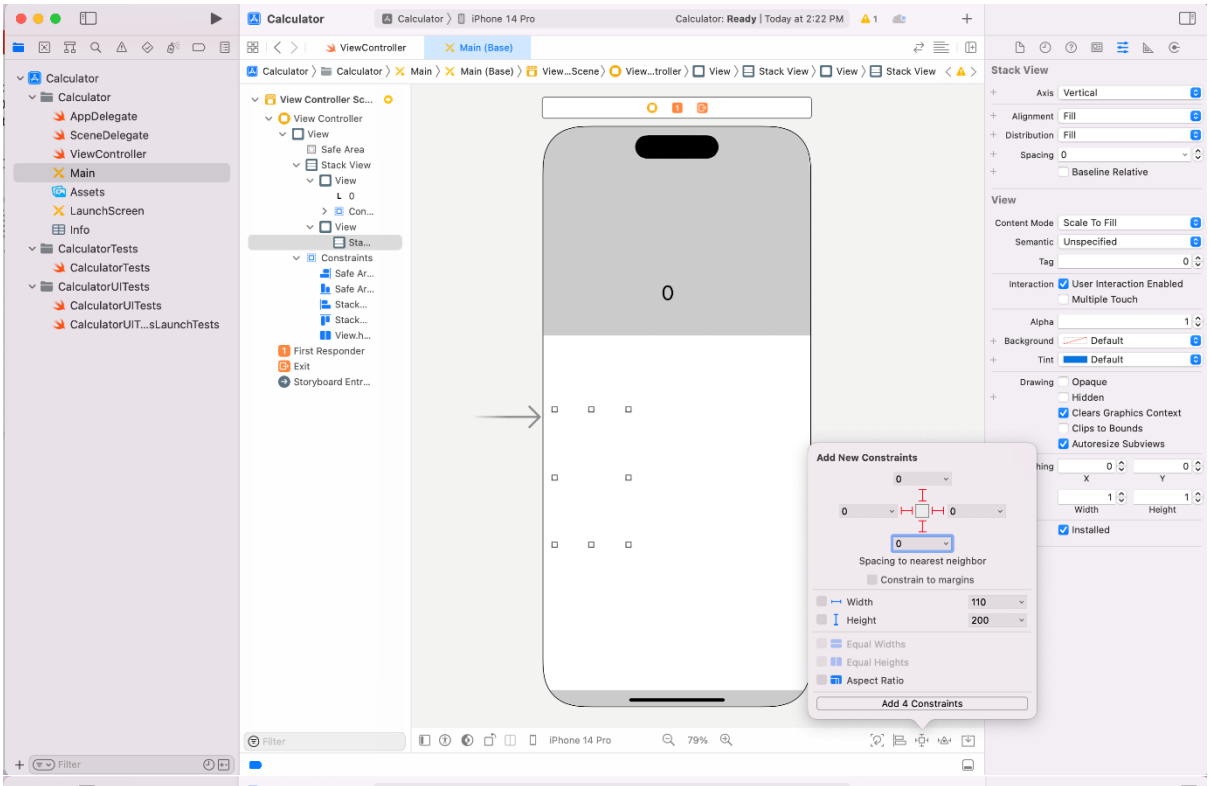


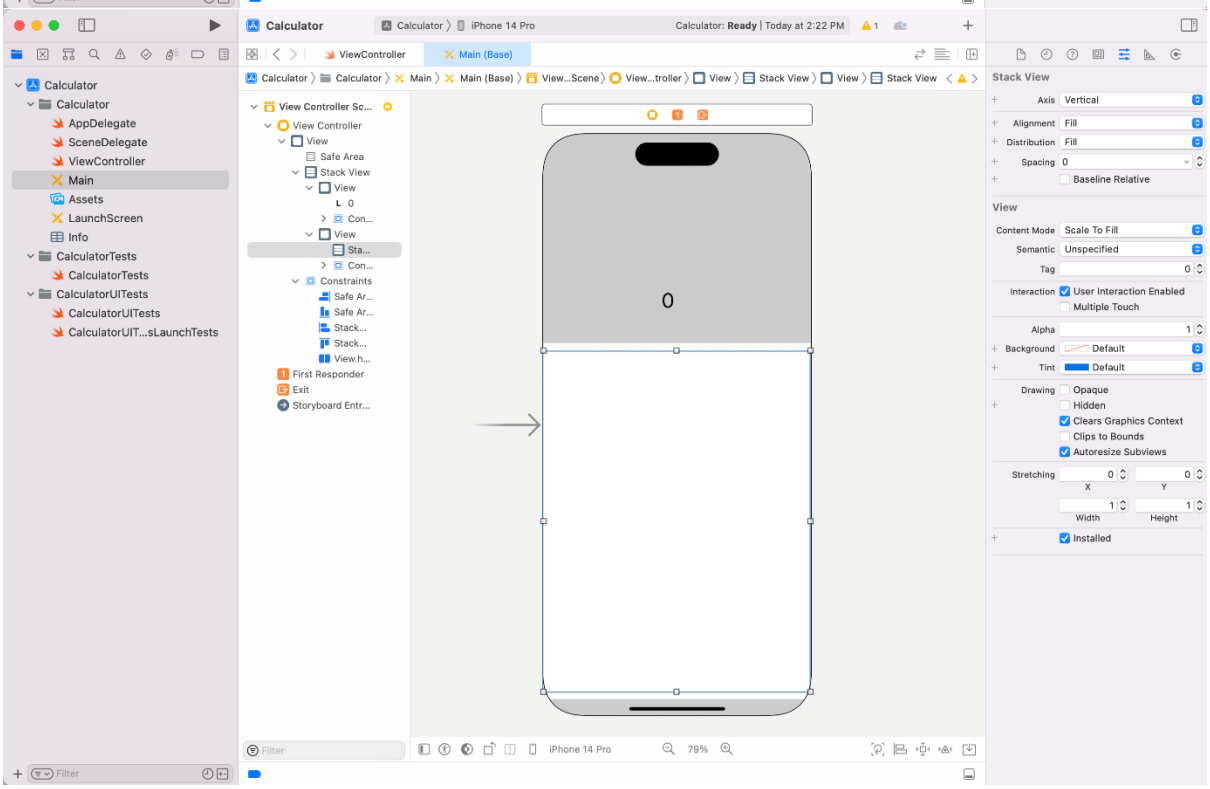

Calculator
Calculator: Ready | Today at 2:08 PM
Main (Base)
ViewController
View Controller Scene
View Controller
First Responder
Exit
Storyboard Entry Point
Safe Area
Stack View
Content Mode
Scale To Fill
Semantic
Unspecified
Interaction
User Interaction Enabled
Multiple Touch
Alpha
Background
System Background...
Tint
Default
Drawing
Opaque
Hidden
Clears Graphics Context
Clips to Bounds
Autoresize Subviews
Stretching
Installed

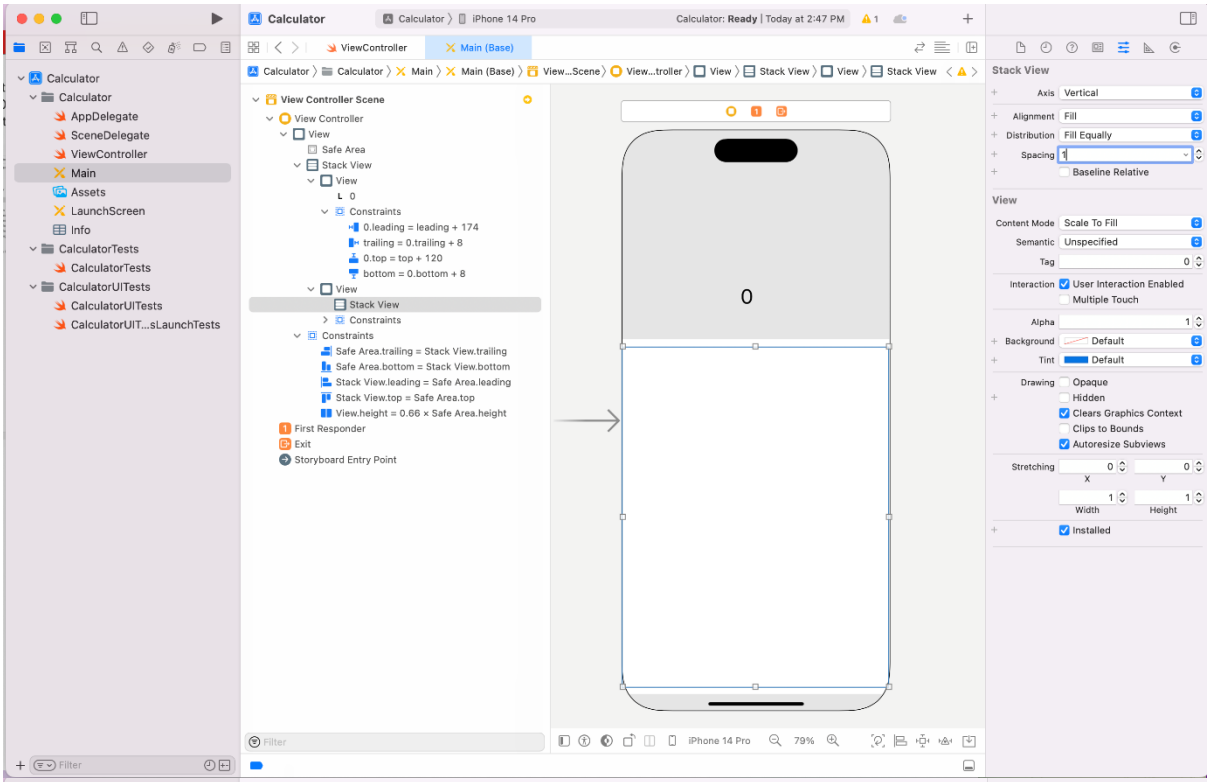

Calculator
Calculator: Ready | Today at 2:08 PM
Main (Base)
ViewController
View Controller Scene
View Controller
Leading Space to Safe Area
Top Space to Safe Area
Trailing Space to Safe Area
Bottom Space to Safe Area
Center Horizontally in Safe Area
Center Vertically in Safe Area
Equal Widths
Equal Heights
Aspect Ratio
Hold Shift to select multiple
Hold Option for alternates
No Selection

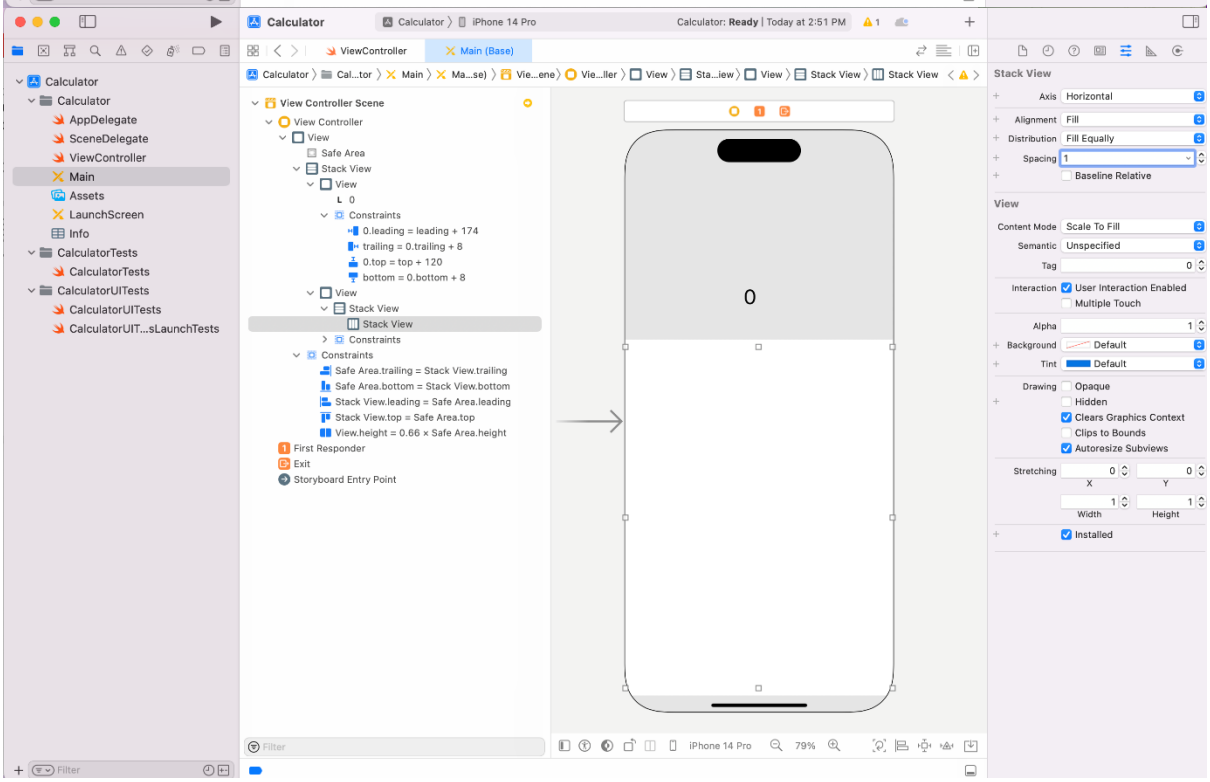


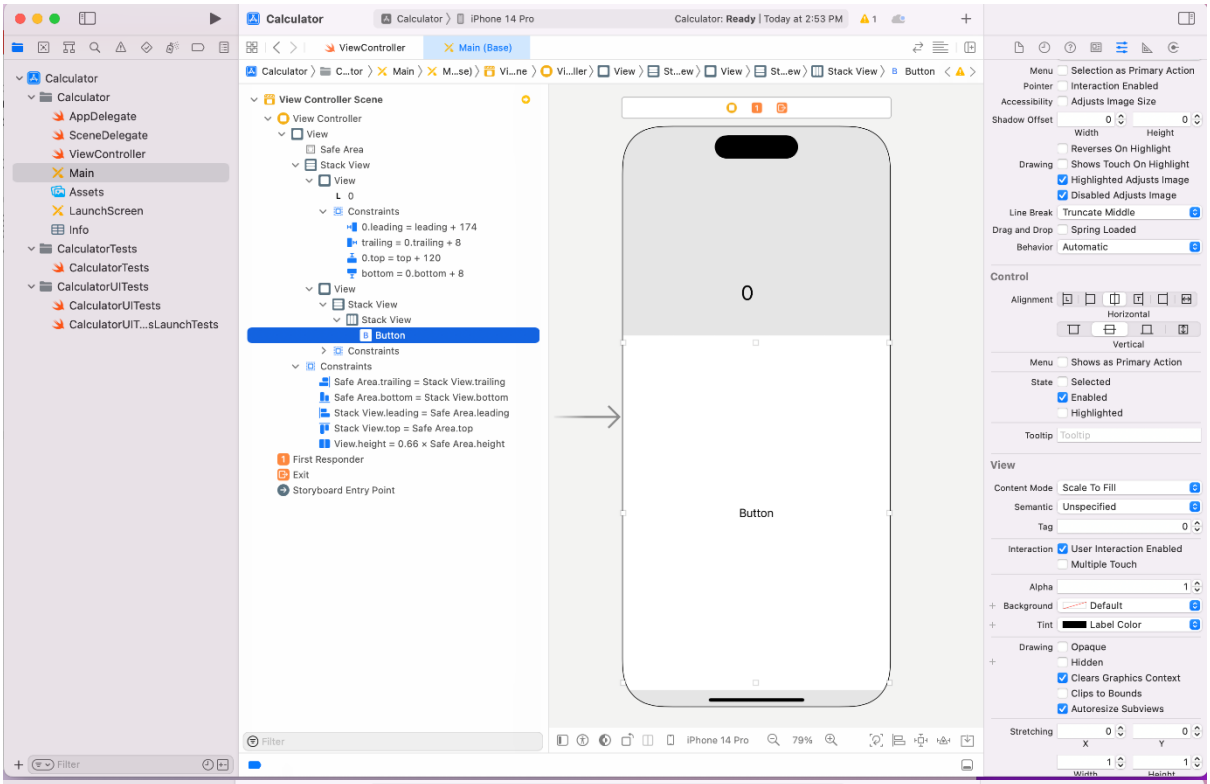
Exercise - Optionals

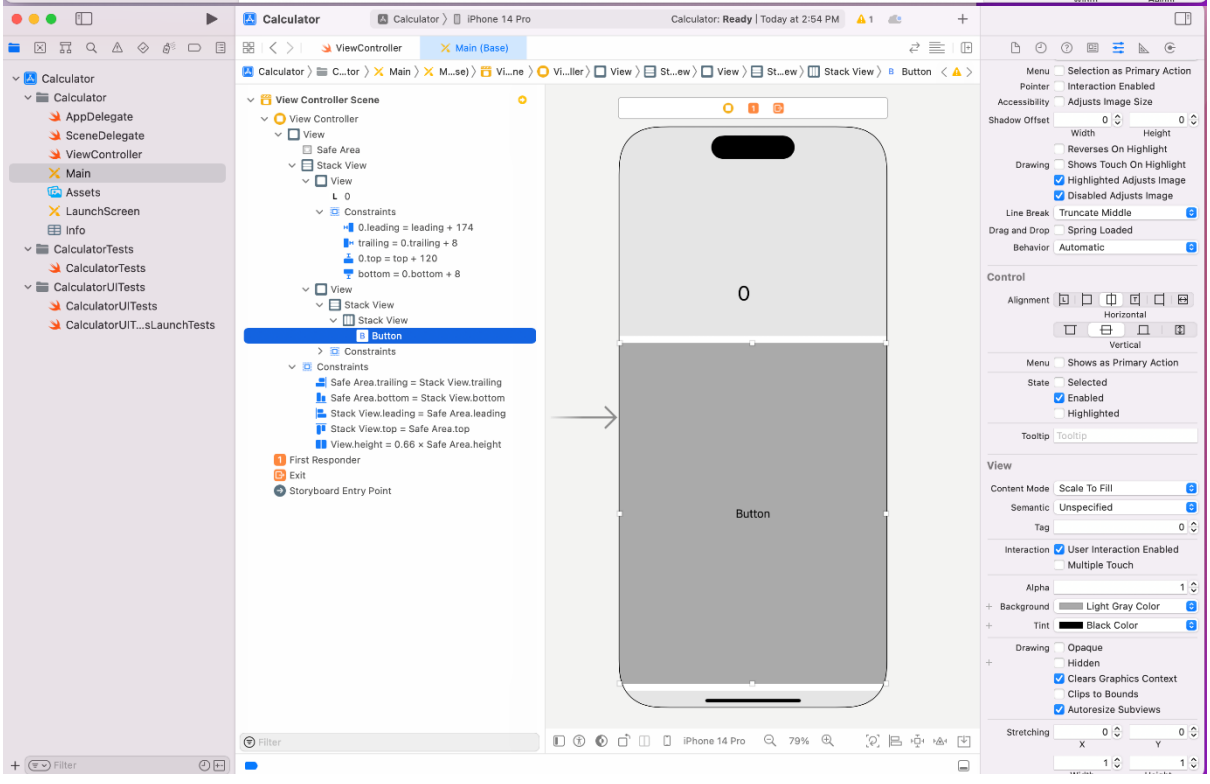

Calculator
Calculator: Ready | Today at 2:22 PM
Add New Constraints
Spacing to nearest neighbor
Constrain to margins
Width
Height
Equal Widths
Equal Heights
Aspect Ratio
Add 4 Constraints
Stack View
Axis
Vertical
Alignment
Fill
Distribution
Fill
Spacing
Baseline Relative
View
Content Mode
Scale To Fill
Semantic
Unspecified
Tag
Interaction
User Interaction Enabled
Multiple Touch
Alpha
Background
Default
Tint
Default
Drawing
Opaque
Hidden
Clears Graphics Context
Clips to Bounds
Autoresize Subviews
Installed

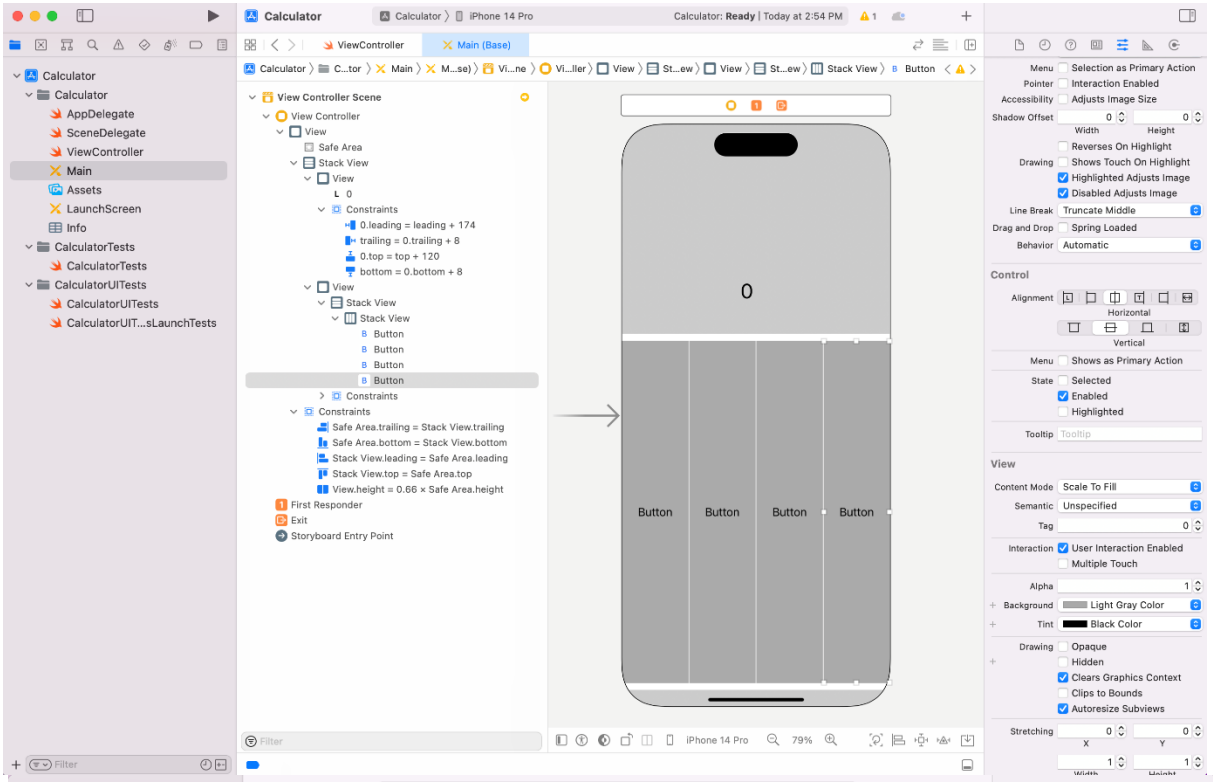

Calculator
Calculator: Ready | Today at 2:22 PM
Stack View
Axis
Vertical
Alignment
Fill
Distribution
Fill
Spacing
Baseline Relative
View
Content Mode
Scale To Fill
Semantic
Unspecified
Tag
Interaction
User Interaction Enabled
Multiple Touch
Alpha
Background
Default
Tint
Default
Drawing
Opaque
Hidden
Clears Graphics Context
Clips to Bounds
Autoresize Subviews
Stretching
Installed

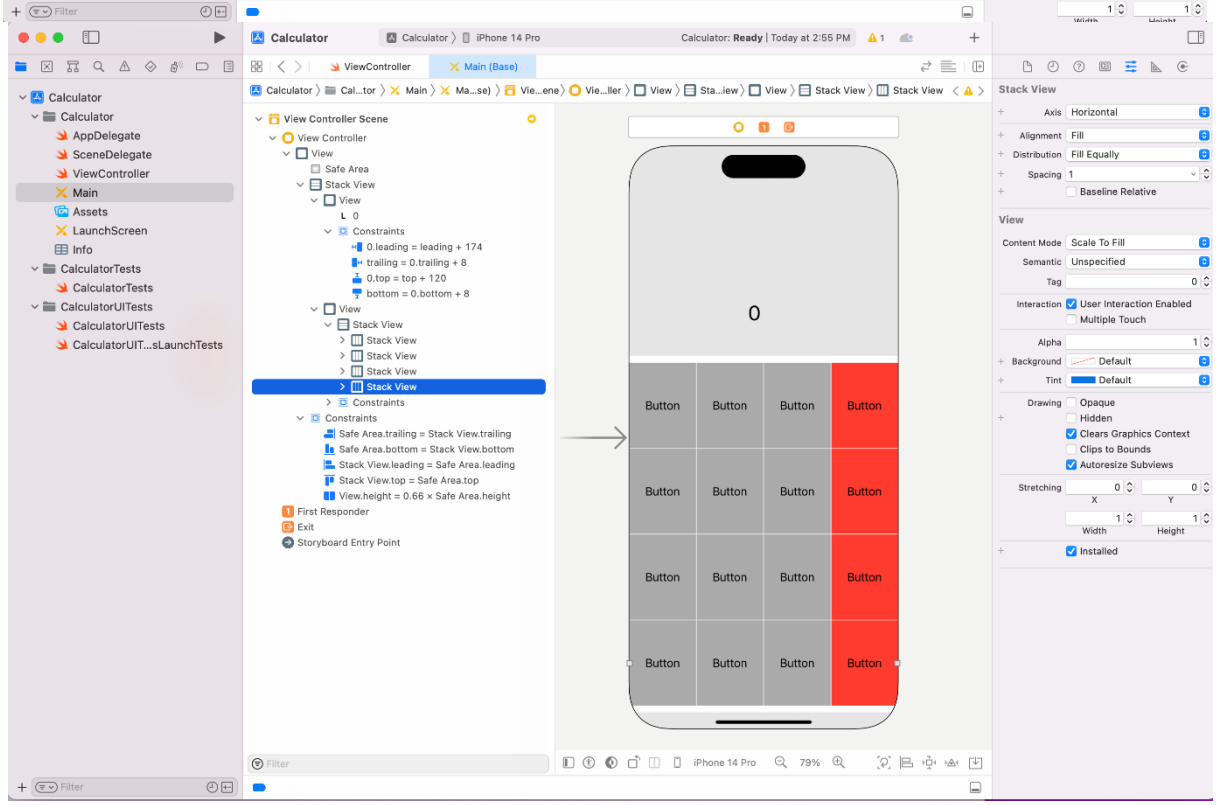

Calculator
Calculator: Ready | Today at 2:47 PM
ViewController
Main (Base)
View Controller Scene
0
Stack View
Axis Vertical
Alignment Fill
Distribution Fill Equally
Spacing 1
Baseline Relative
Content Mode Scale To Fill
Semantic Unspecified
Interaction User Interaction Enabled
Multiple Touch
Alpha
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clips to Bounds
Autoresize Subviews
Stretching
Installed

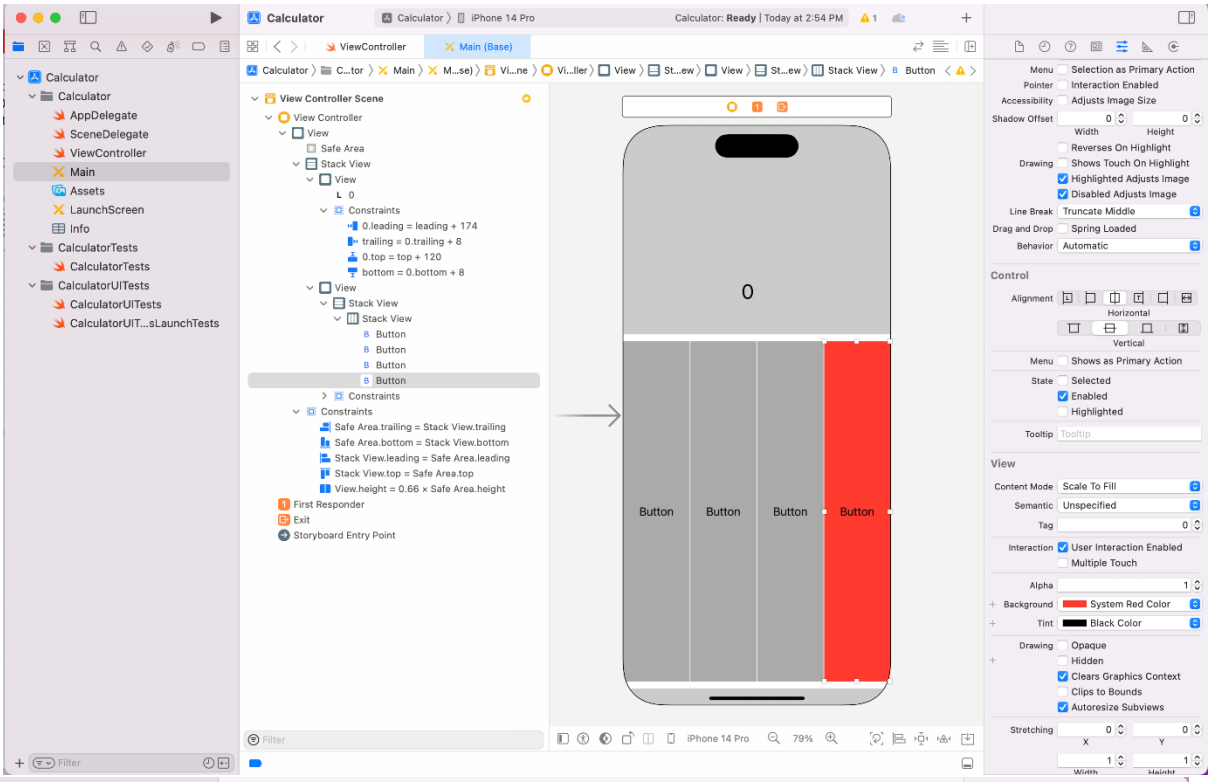

Calculator
Calculator: Ready | Today at 2:51 PM
ViewController
Main (Base)
View Controller Scene
0
Stack View
Axis Horizontal
Alignment Fill
Distribution Fill Equally
Spacing 1
Baseline Relative
Content Mode Scale To Fill
Semantic Unspecified
Interaction User Interaction Enabled
Multiple Touch
Alpha
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clips to Bounds
Autoresize Subviews
Stretching
Installed

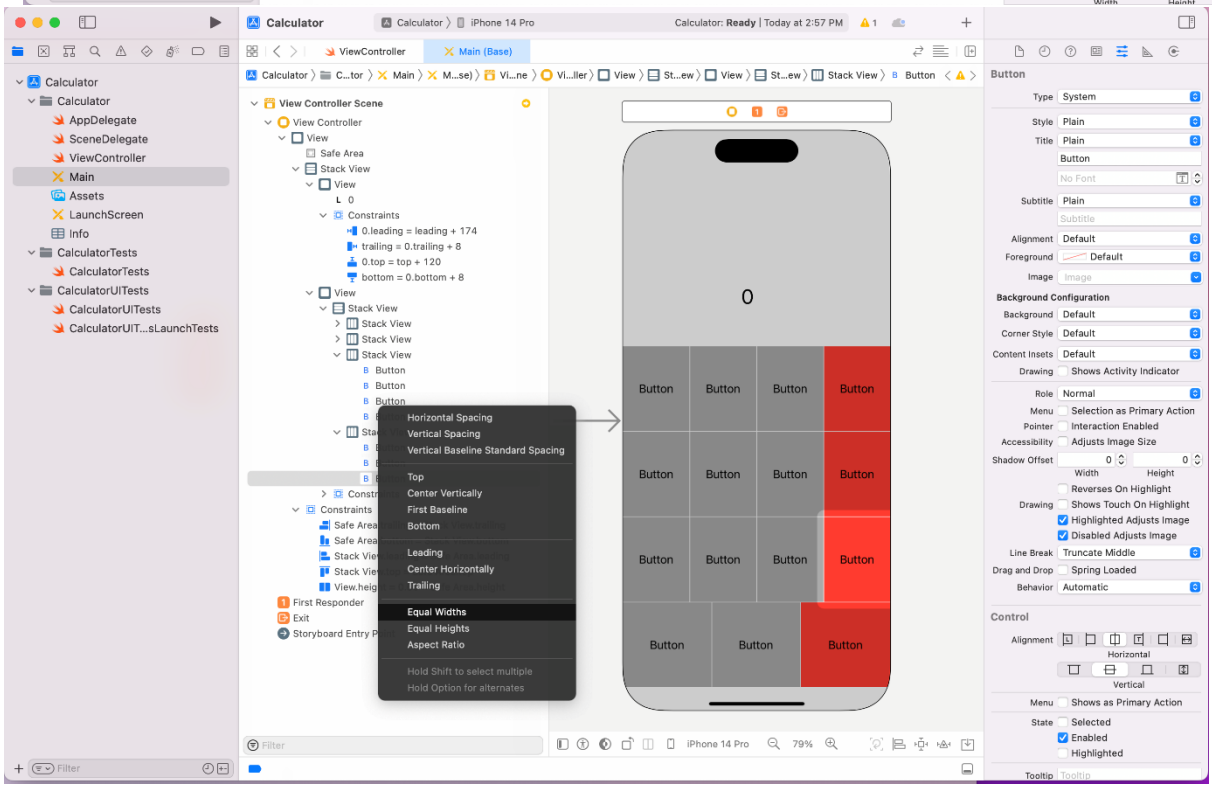

Calculator
Calculator: Ready | Today at 2:53 PM
ViewController
Main (Base)
View Controller Scene
View Controller
View
Safe Area
Stack View
Constraints
Button
First Responder
Exit
Storyboard Entry Point
0
Button
iPhone 14 Pro
79%

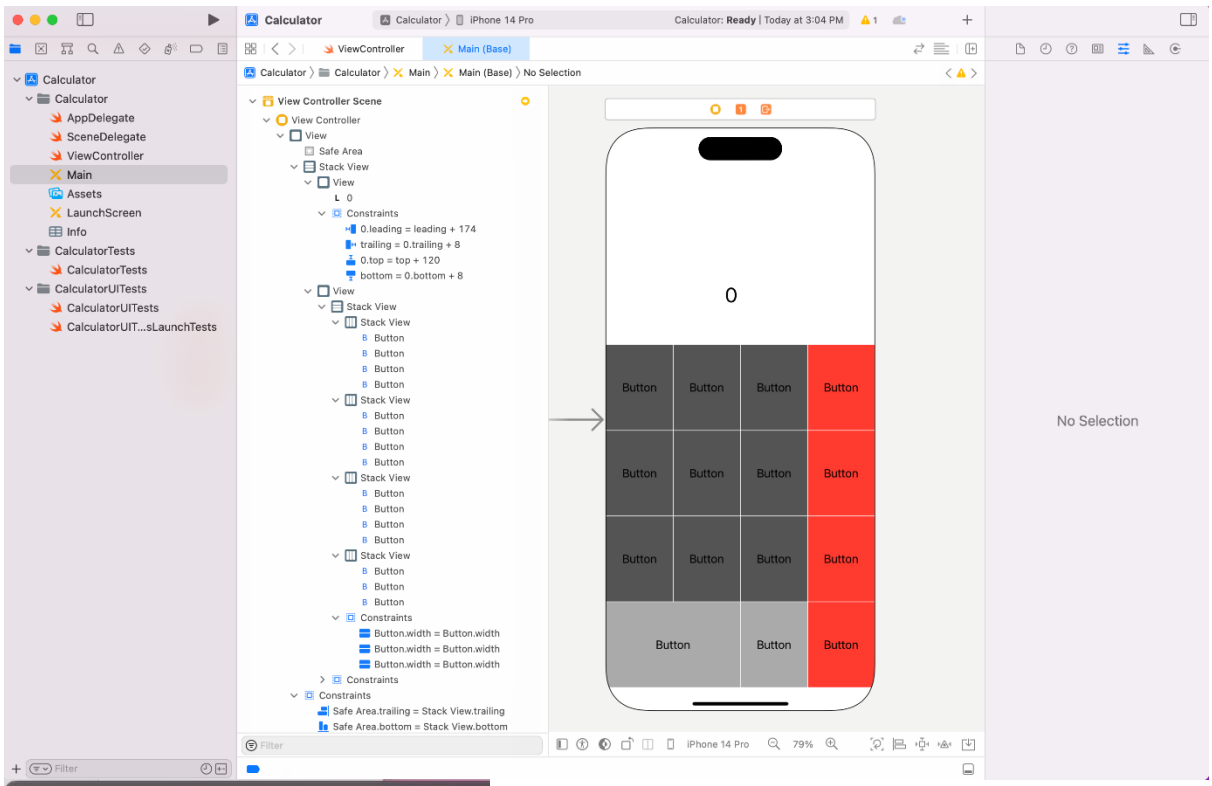

Calculator
Calculator: Ready | Today at 2:54 PM
ViewController
Main (Base)
View Controller Scene
View Controller
View
Safe Area
Stack View
Constraints
Button
First Responder
Exit
Storyboard Entry Point
0
Button
iPhone 14 Pro
79%

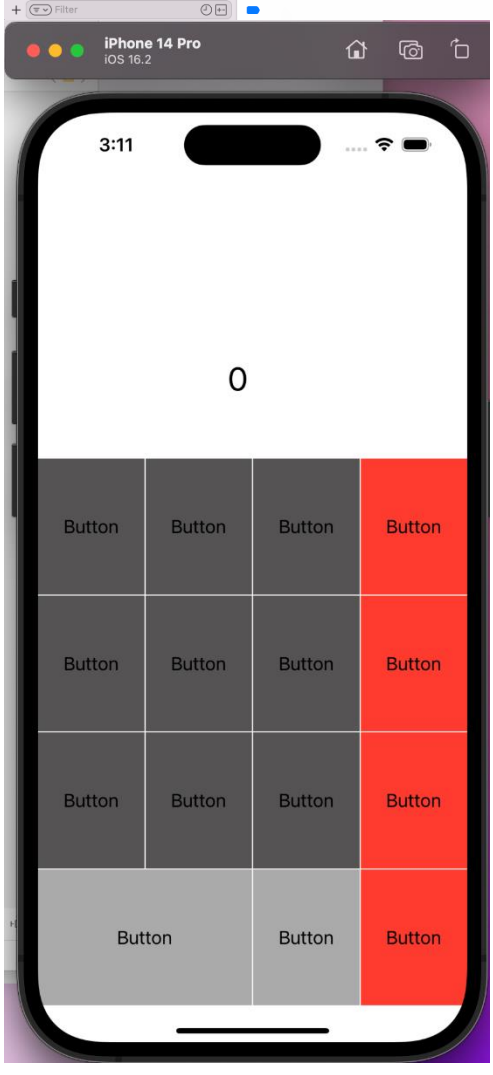

Calculator
Calculator: Ready | Today at 2:54 PM
View Controller Scene
0
Button
Button
Button
Button


Calculator
Calculator: Ready | Today at 2:55 PM
View Controller Scene
0
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button


Calculator
Calculator: Ready | Today at 2:54 PM
ViewController
Main (Base)
View Controller Scene
0
Button
Button
Button
Button

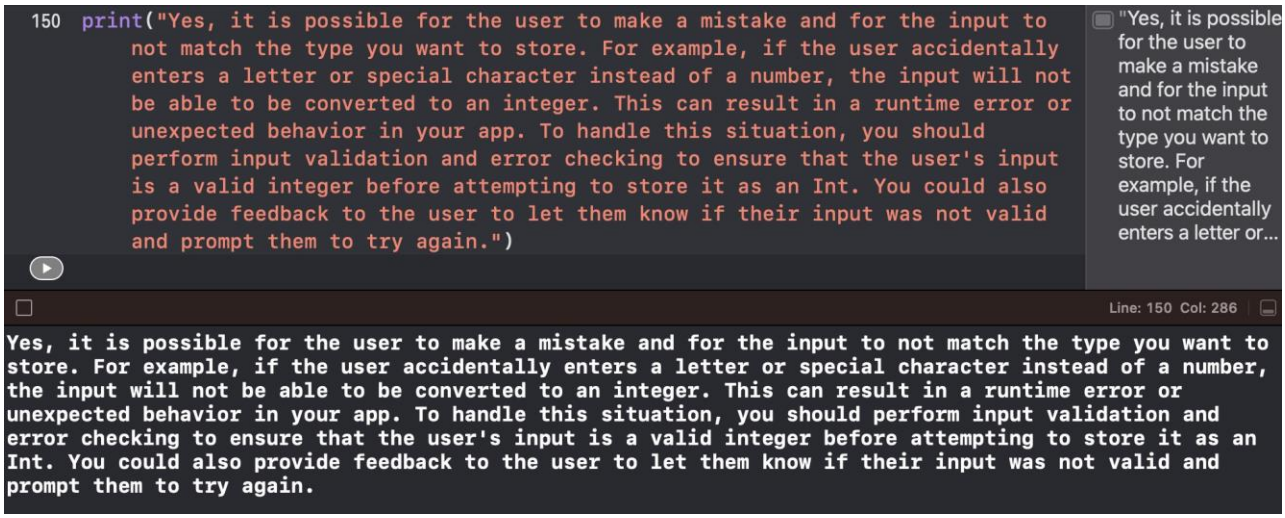

Calculator
Calculator: Ready | Today at 2:57 PM
ViewController
Main (Base)
View Controller Scene
Horizontal Spacing
Vertical Spacing
Vertical Baseline Standard Spacing
Top
Center Vertically
First Baseline
Bottom
Leading
Center Horizontally
Trailing
Equal Widths
Equal Heights
Aspect Ratio
Hold Shift to select multiple
Hold Option for alternates
0
Button


Calculator
Calculator: Ready | Today at 3:04 PM
ViewController
Main (Base)
No Selection
0
Button
Stack View
Constraints

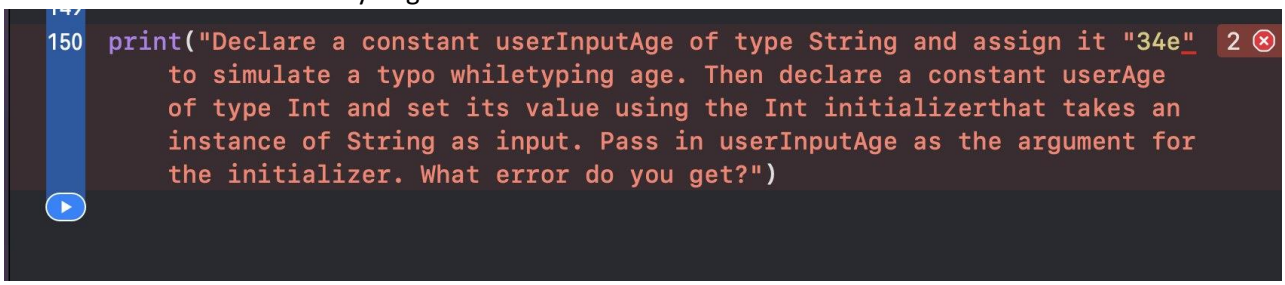

iPhone 14 Pro
iOS 16.2
3:11
0
Button

2. Throughout the exercises in this playground, you will be printing optional values. The Swift compiler will display a warning: "Expression implicitly coerced from `Int?` to Any." For the purposes of these exercises, you can ignore this warning.

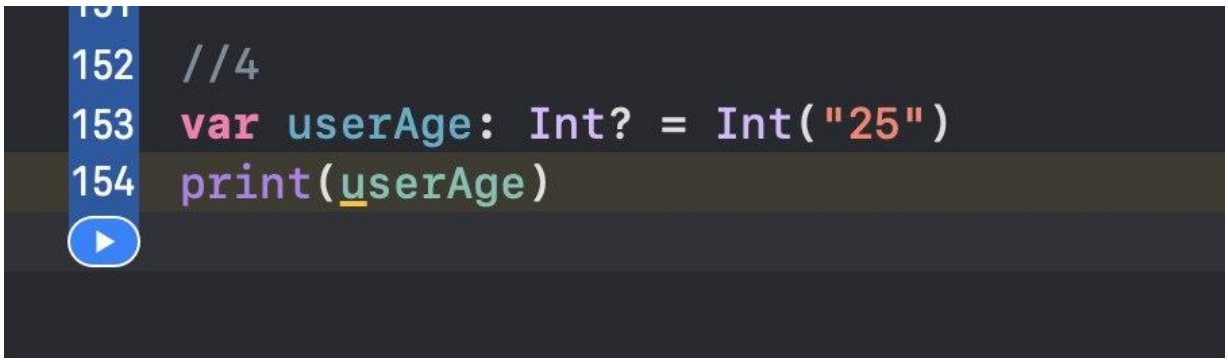
Imagine you have an app that asks the user to enter his/her age using the keyboard. When your app allows a user to input text, what is captured for you is given as a `String`. However, you want to store this information as an `Int`. Is it possible for the user to make a mistake and for the input to not match the type you want to store?



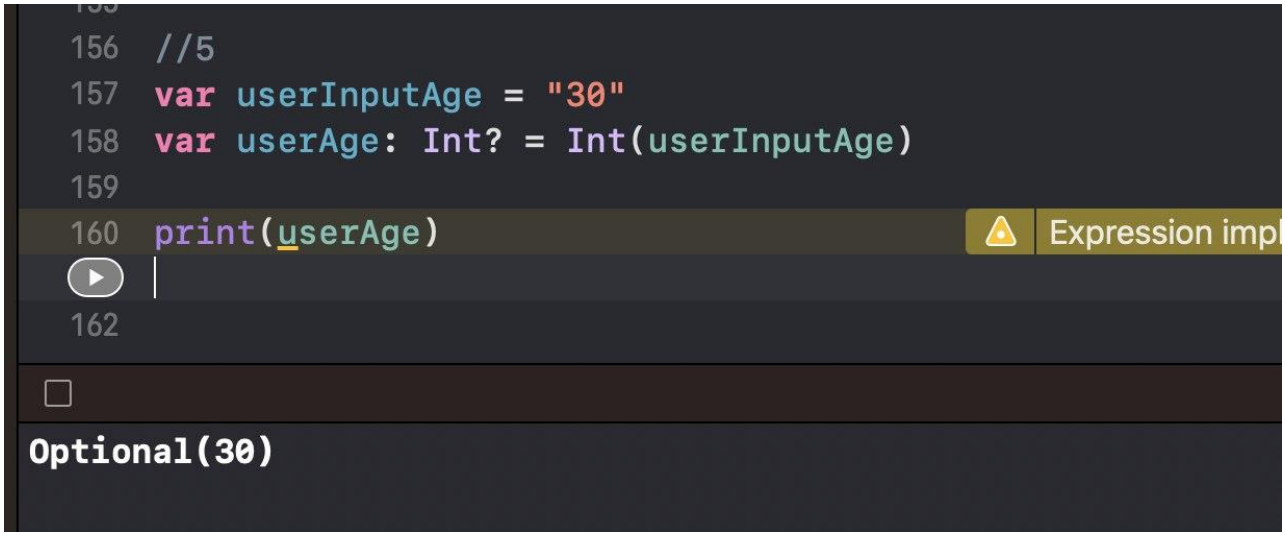
3. Declare a constant `userInputAge` of type `String` and assign it "34e" to simulate a typo while typing age. Then declare a constant `userAge` of type `Int` and set its value using the `Int` initializer that takes an instance of `String` as input. Pass in `userInputAge` as the argument for the initializer. What error do you get?



4. Go back and change the type of `userAge` to `Int?`, and print the value of `userAge`. Why is `userAge`'s value `nil`? Provide your answer in a comment or print statement below.



5. Now go back and fix the typo on the value of `userInputAge`. Is there anything about the value printed that seems off? Print `userAge` again, but this time unwrap `userAge` using the force unwrap operator.



6. Now use optional binding to unwrap `userAge`. If `userAge` has a value, print it to the console.

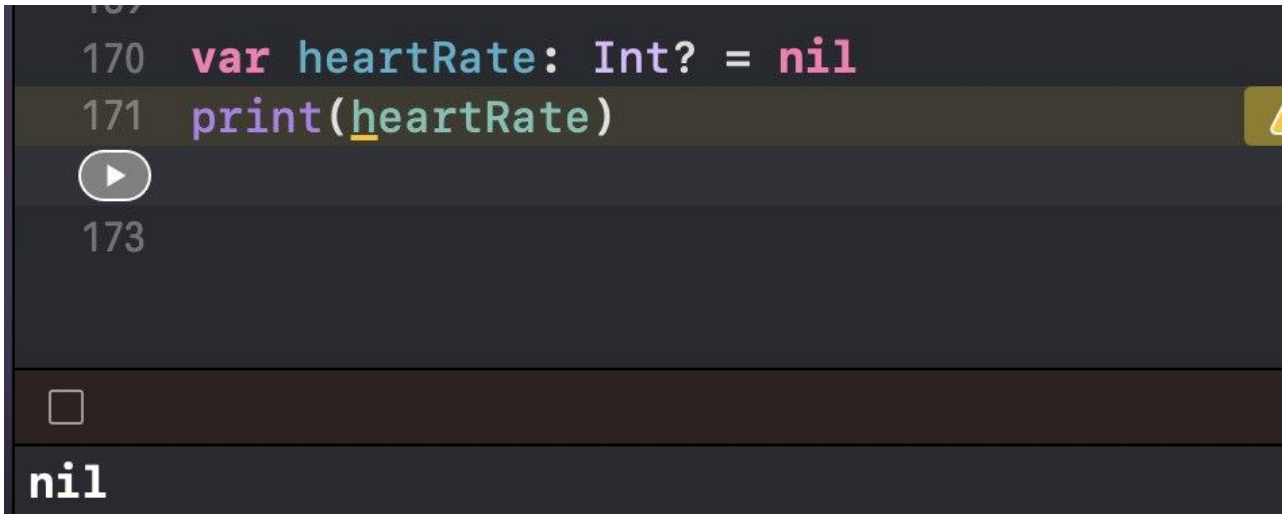
```swift
//6
var userInputAge = "30"
var userAge: Int? = Int(userInputAge)

if let unwrappedAge = userAge {
    print(unwrappedAge)
}
```

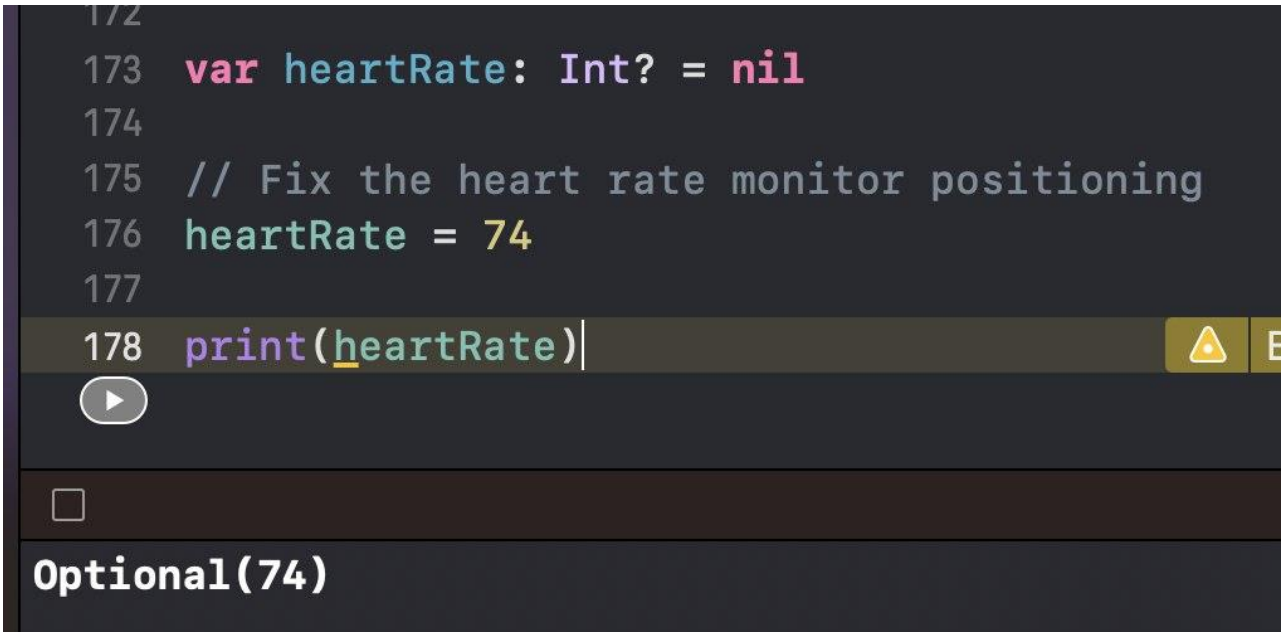
```
30
```

App Exercise - Finding a Heart Rate

7. Many APIs that give you information gathered by the hardware return optionals. For example, an API for working with a heart rate monitor may give you `nil` if the heart rate monitor is adjusted poorly and cannot properly read the user's heart rate. Declare a variable `heartRate` of type `Int?` and set it to `nil`. Print the value.



8. In this example, if the user fixes the positioning of the heart rate monitor, the app may get a proper heart rate reading. Below, update the value of `heartRate` to 74. Print the value.



```
173 var heartRate: Int? = nil
174
175 // Fix the heart rate monitor positioning
176 heartRate = 74
177
178 print(heartRate)
```

```
Optional(74)
```

9. As you've done in other app exercises, create a variable `hrAverage` of type `Int` and use the values stored below and the value of `heartRate` to calculate an average heart rate.

let oldHR1 = 80

let oldHR2 = 76

let oldHR3 = 79

let oldHR4 = 70

```
180 //9
181 let oldHR1 = 80
182 let oldHR2 = 76
183 let oldHR3 = 79
184 let oldHR4 = 70
185 var heartRate: Int? = 74
186
187 // Calculate the average heart rate
188 let hrAverage = (oldHR1 + oldHR2 + oldHR3 + oldHR4 + (heartRate ?? 0)) / 5
189
190 print(hrAverage)
```

```
75
```

10. If you didn't unwrap the value of `heartRate`, you've probably noticed that you cannot perform mathematical operations on an optional value. You will first need to unwrap `heartRate`.

Safely unwrap the value of `heartRate` using optional binding. If it has a value, calculate the average heart rate using that value and the older heart rates stored above. If it doesn't have a value, calculate the average heart rate using only the older heart rates. In each case, print the value of `hrAverage`.

```
193 //10
194 let oldHR1 = 80
195 let oldHR2 = 76
196 let oldHR3 = 79
197 let oldHR4 = 70
198 var heartRate: Int? = 74
199
200 // Calculate the average heart rate
201 var hrSum = oldHR1 + oldHR2 + oldHR3 + oldHR4
202 var hrCount = 4
203
204 if let currentHR = heartRate {
205     hrSum += currentHR
206     hrCount += 1
207 }
208
209 let hrAverage = hrSum / hrCount
210
211 print(hrAverage)
```

```
75
```

Type Casting and Inspection

11. Create a collection of type [Any], including a few doubles, integers, strings, and booleans within the collection. Print the contents of the collection.

```swift
//11
let mixedCollection: [Any] = [42, "hello", 3.14, true, 27.5, false, 100]

print(mixedCollection)
```

```
[42, "hello", 3.14, true, 27.5, false, 100]
```

12. Loop through the collection. For each integer, print "The integer has a value of ", followed by the integer value. Repeat the steps for doubles, strings and booleans.

```swift
//12
let mixedCollection: [Any] = [42, "hello", 3.14, true, 27.5, false, 100]

for element in mixedCollection {
    if let integer = element as? Int {
        print("The integer has a value of", integer)
    } else if let double = element as? Double {
        print("The double has a value of", double)
    } else if let string = element as? String {
        print("The string has a value of", string)
    } else if let bool = element as? Bool {
        print("The boolean has a value of", bool)
    }
}
```

```
The integer has a value of 42
The string has a value of hello
The double has a value of 3.14
The boolean has a value of true
The double has a value of 27.5
The boolean has a value of false
The integer has a value of 100
```

13. Create a [String : Any] dictionary, where the values are a mixture of doubles, integers, strings, and booleans. Print the key/value pairs within the collection

```swift

//13
let myDictionary: [String: Any] = ["name": "Alice", "age": 25, "height": 1.65,
    "isStudent": true]

for (key, value) in myDictionary {
    print("\(key): \(value)")
}
```

```
isStudent: true
name: Alice
age: 25
height: 1.65
```

14. Create a variable `total` of type `Double` set to 0. Then loop through the dictionary, and add the value of each integer and double to your variable's value. For each string value, add 1 to the total. For each boolean, add 2 to the total if the boolean is `true`, or subtract 3 if it's `false`. Print the value of `total`.

```swift
//14
let myDictionary: [String: Any] = ["name": "Alice", "age": 25, "height": 1.65, "isStudent": true]

var total: Double = 0

for (_, value) in myDictionary {
    if let intValue = value as? Int {
        total += Double(intValue)
    } else if let doubleValue = value as? Double {
        total += doubleValue
    } else if value is String {
        total += 1
    } else if let boolValue = value as? Bool {
        if boolValue {
            total += 2
        } else {
            total -= 3
        }
    }
}

print(total)
```

```
29.65
```

15. Create a variable `total2` of type `Double` set to 0. Loop through the collection again, adding up all the integers and doubles. For each string that you come across during the loop, attempt to convert the string into a number, and add that value to the total. Ignore booleans. Print the total.

```swift
//15
let collection: [Any] = [1, 3.5, "2", true, "3.14", false, 5]

var total2: Double = 0

for item in collection {
    if let number = item as? Double {
        total2 += number
    } else if let number = item as? Int {
        total2 += Double(number)
    } else if let string = item as? String, let number = Double(string) {
        total2 += number
    }
}

print(total2)
```

```
14.64
```

App Exercise - Workout Types

16. You fitness tracking app may allow users to track different kinds of workouts. When architecting the app, you may decide to have a `Workout` base class from which other types of workout classes inherit. Below are three classes. `Workout` is the base class with `time` and `distance` properties, and `Run` and `Swim` are subclasses that add more specific properties to the `Workout` class. Also provided is a `workouts` array that represents a log of past workouts. You'll use these classes and the array for the exercises below.

```
class  Workout  {
  let time: Double
  let distance: Double

  init(time: Double, distance: Double) {
    self.time = time
    self.distance = distance
  }
```

```
}

 class Run: Workout {
    let cadence: Double

    init(cadence: Double, time: Double, distance: Double) {
       self.cadence = cadence
       super.init(time: time, distance: distance)
    }
 }

 class Swim: Workout {
    let stroke: String

    init(stroke: String, time: Double, distance: Double) {
       self.stroke = stroke
       super.init(time: time, distance: distance)
    }
 }

 var workouts: [Workout] = [
    Run(cadence: 80, time: 1200, distance: 4000),
    Swim(stroke: "Freestyle", time: 32.1, distance: 50),
    Swim(stroke: "Butterfly", time: 36.8, distance: 50),
    Swim(stroke: "Freestyle", time: 523.6, distance: 500),
    Run(cadence: 90, time: 358.9, distance: 1600)
 ]
```

17. Write simple functions called `describeRun(runningWorkout:)` and `describeSwim(swimmingWorkout:)` that take a `Run` object and a `Swim` object, respectively. Neither should return values. Each function should print a description of the workout, including the run's cadence or the swim's stroke. Time is represented in seconds, distance is represented in meters, and cadence is represented in steps per minute.

```
281 //17
282 func describeRun(runningWorkout: Run) {          Cannot find type 'Run' in scope
283     print("A running workout with a cadence of \(runningWorkout.cadence) steps
        per minute. Time: \(runningWorkout.time) seconds. Distance:
        \(runningWorkout.distance) meters.")
284 }
285
286 func describeSwim(swimmingWorkout: Swim) {       Cannot find type 'Swim' in scope
287     print("A swimming workout with a stroke of \(swimmingWorkout.stroke). Time:
        \(swimmingWorkout.time) seconds. Distance: \(swimmingWorkout.distance)
        meters.")
288 }
```

```
func describeRun(runningWorkout: Run) {
                                 ^~~

error: 2020btecs00037.playground:285:36: error: cannot find type 'Swim' in scope
func describeSwim(swimmingWorkout: Swim) {
                                   ^~~~
```

18. Now loop through each workout in `workouts` and, using type casting, call either `describeRun(runningWorkout:)` or `describeSwim(swimmingWorkout:)` on each. Observe what is printed to the console.

```
289
290 for workout in workouts {
291     if let run = workout as? Run {
292         describeRun(runningWorkout: run)
293     } else if let swim = workout as? Swim {
294         describeSwim(swimmingWorkout: swim)
295     }
296 }
```

****