Swift Playgrounds App Project

Ethan & Tongyu

Swift Accelerator Alumni

Links to slides: tk.sg/sis23-playgroundsappdev

Welcome!

Have you ever wondered how the apps on your devices are made? In this workshop, you'll learn how to use Swift Playgrounds & SwiftUI to build your very first app: a fun and engaging game of trivia!

Who are we?

- **Ethan**
  - Swift Accelerator batch of 2021 and created ArrivalSG
  - Graduated from the *School of Science and Technology, Singapore* in 2021
- **Tongyu**
  - Swift Accelerator batch of 2020 — created app Habitat
  - Graduated from *Raffles Girls' School* in 2022 :)

How to make an app?

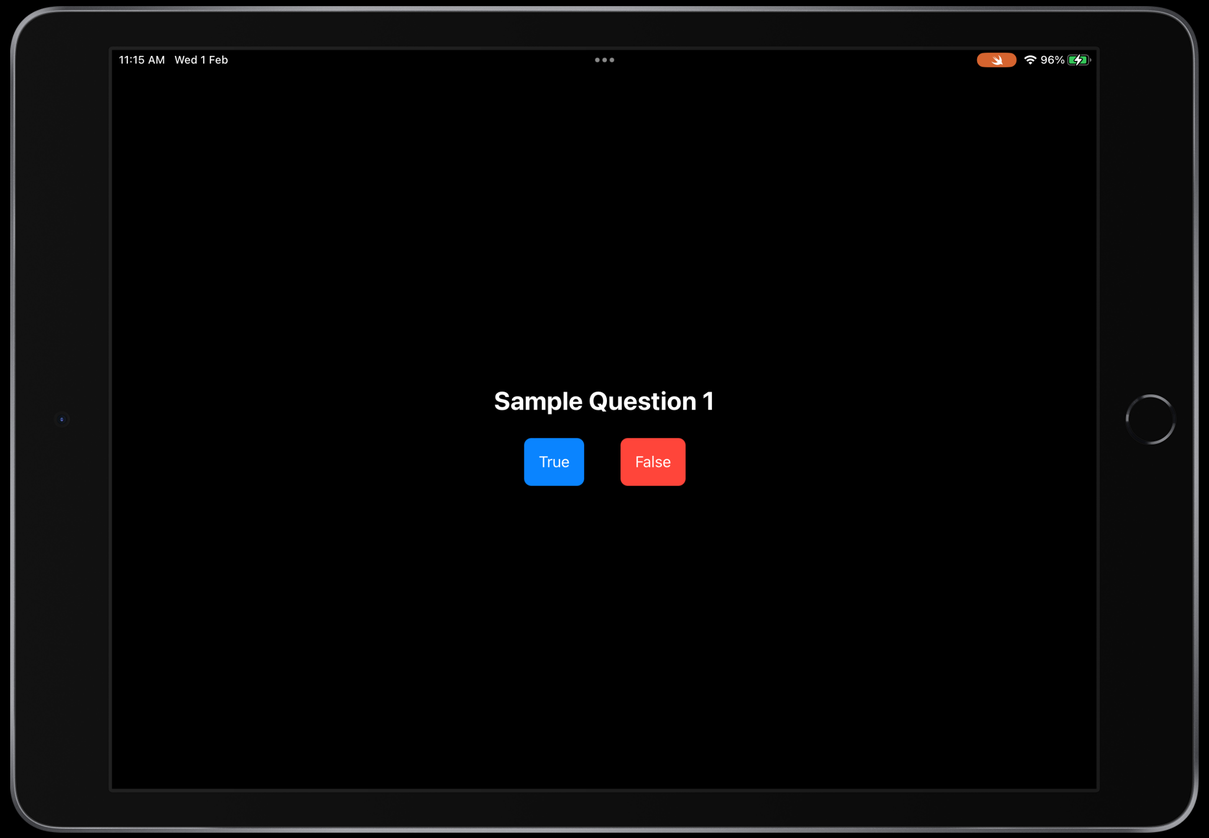
- The process of making an app can be roughly split into two parts — how the app looks, and how the app works.
- Using Swift Playgrounds, we can do both, with two languages: **Swift** and **SwiftUI**.
- We need to define our app's **layout**, and then put in the code to make it work.

What is Swift & SwiftUI?

- **Swift** is the Programming Language which is used to create apps for Apple devices. It is the *logic* code that tells the app what to do.
- **SwiftUI** is a way to build User Interfaces on Apple devices using Swift. This is the *layout* code that tells the app what to look like.
- Swift != SwiftUI

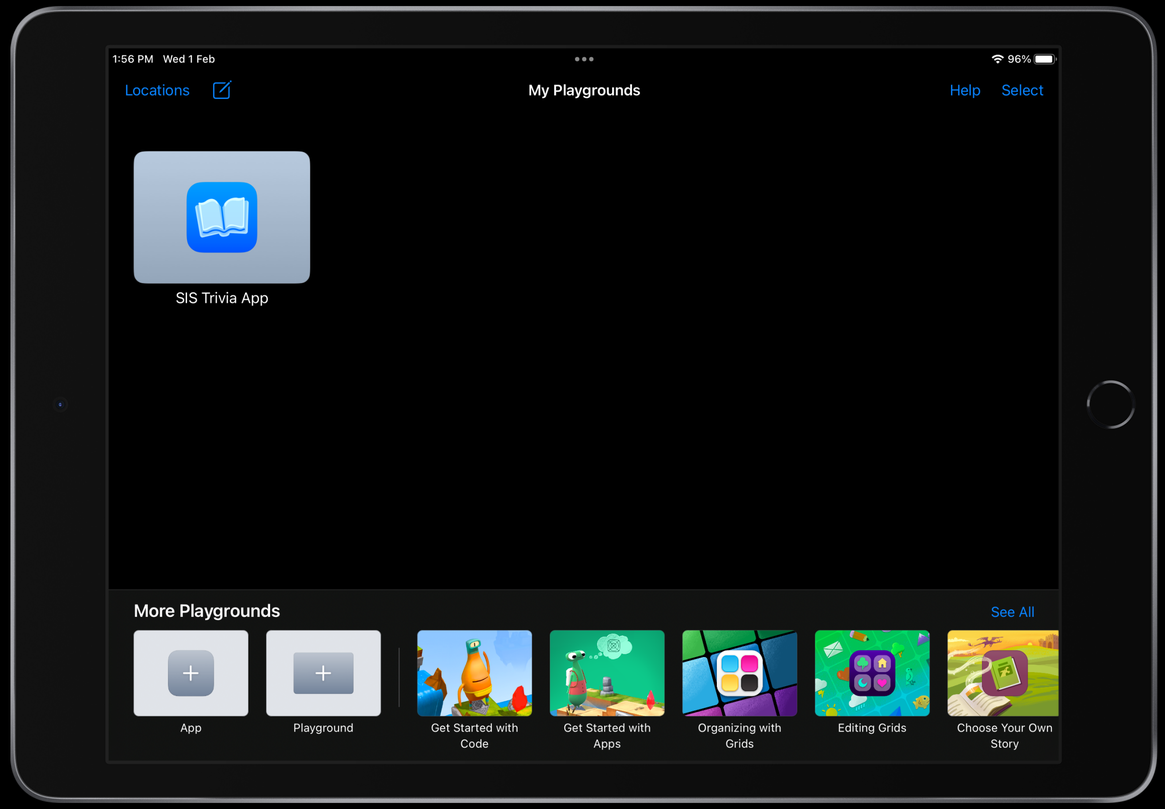
Today's agenda

- We'll be using Swift and SwiftUI to create a trivia app with true-or-false questions.
- Think of some fun trivia questions to ask!



Let's begin!

- Open Swift Playgrounds.
- You'll find an app named **SIS Trivia App**. Tap on it to open the template that we have prepared.
- We will be editing **ContentView.swift** for our app. Do not edit the other files!

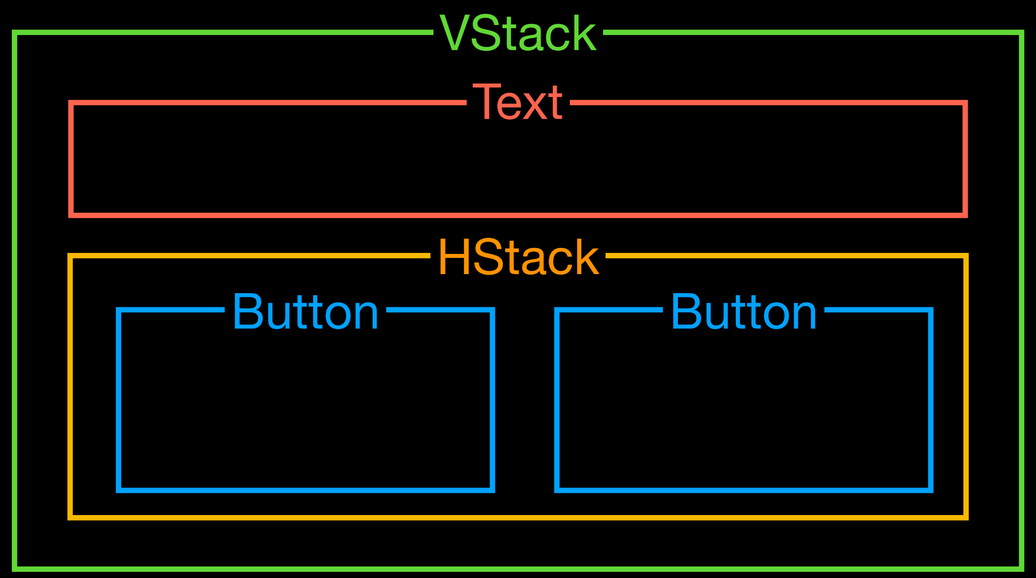


Layout

How your app will look like.

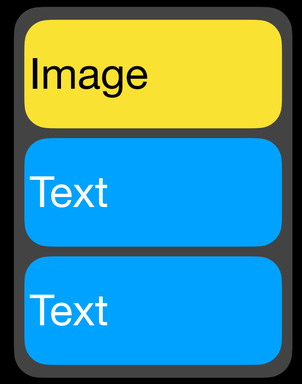
Laying it out

- Usually, before starting to code an app, we create a simple design or a *wireframe* for it.
  - This helps us understand how our user interface will look like beforehand.

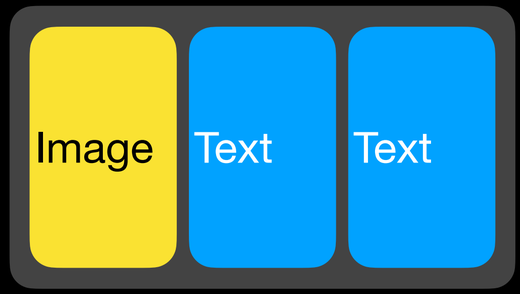


Views & Stacks

- A **view** is a piece of you app's UI — everything, from the buttons to text to stacks, are a view.
- Stacks are ways to arrange multiple views into a single view.
  - Think of it as organising your views, like a shelf.
- VStack: aligns objects vertically.
- HStack: aligns objects horizontally.



VStack



HStack

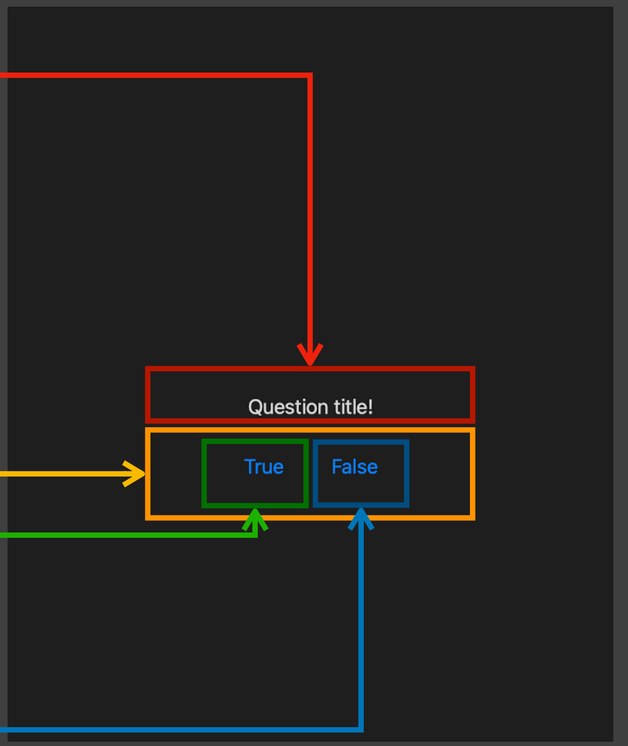
Layout explained

```swift
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Question title!")

            HStack {
                Button("True") { // The 'true' option
                    // Code that runs when the button is pressed
                }

                Button("False") { // The 'false' option
                    // Code that runs when the button is pressed
                }
            }
        }
    }
}
```

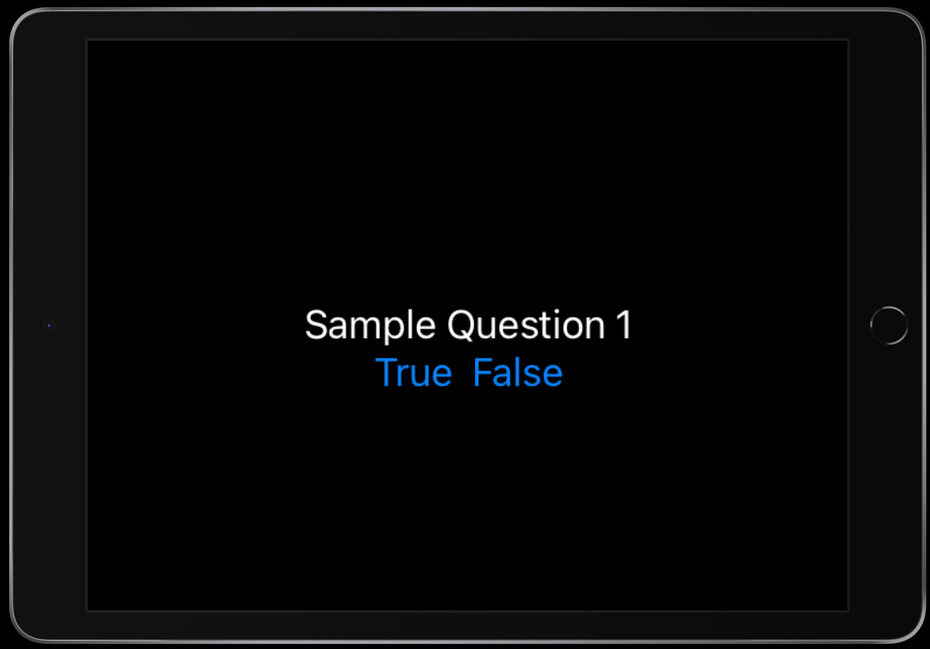
Preview



ContentView.swift

Checkpoint!

- At this point, you should be able to see a screen similar to this — a question title, and two buttons.
- Raise your hand if you need help!



Adding Data

How do we have a trivia app without any questions??

Variables and Constants

- It’s like a named storage space for a value.
  - Think of it as a box that can store different kinds of items. A variable can store different types of data.
  - To create a variable, we use a **keyword**, followed by the **variable name**.
- `var` variablename = "some value!"
  - A value that can be changed while the program is running
- `let` constantname = 51947
  - A value that cannot be changed while the program is running
  - It’s value is determined when the constant is created, or, *declared*

Data Types

Different kinds of Data

Huh? What's a Data Type?

- A Data Type is the kind of data that is in a variable.
- In Swift, there are 6 main Data Types:
  String, Int, Float, Double, Bool, and Character
  - Swift also supports creating your OWN Data Types!
- For this workshop, we only be covering String, Int and Bool.

Strings

- Strings are basically lines of text
  - They are denoted by "" in most programming languages
- For example, the following are Strings
  - "Hello World!”
  - "1234"
  - "true"

Integers (Int)

- Integers, or Ints as you will learn in your math lessons, are numbers
  - In Swift, Ints are numbers without any decimal points
- For example, the following are Ints
  - 1234
  - 0
- However,
  - 1.0 is NOT an Int as it contains a decimal point

Booleans (Bool)

- Booleans, or Bools is a true or false value
- In **Swift**, Bools are the following
  - true
  - false
- In languages like Python, Bools can be written as..
  - True
  - False
- Yes, capitalisation DOES matter!

List of Questions

- A ‘list’ in Swift is called an **Array**
  - Arrays are denoted by square brackets [ ].
- What does each question need?
  - The title of the question, or what to ask
  - The correct option — in this case, either **true** or **false**.
- We have created a Structure, also known as a Struct. This can be found in the *Question.swift* file. You don't need to do anything to it!

Adding Questions

- To store our questions, let's use an array of Structs.
  - You can see this on line 12 of ContentView.swift
  - To create a new question, add a comma after the first struct, and type 'Question' and let autocomplete autofill the required information it needs.
  - Then, enter the question's name and correct answer.

```
struct ContentView: View {

    let questions = [Question(name: "Sample Question 1", answer: false), Question(name:
"Sample Question 2", answer: true)]

    . . .
```

ContentView.swift

Take a minute to create as many true or false questions as you want!

We will resume after.

Introducing @State variables

- The word `@State` in front of a variable changes how the variable works.
- It makes it so that if you change something, it will refresh the view to update everything with the new value!
- Add the keyword `@State` before your variable declaration and you can use State Variables.

Showing the Question

- We have our questions ready. How do we show them?
- First, let's create another @State variable, and call it **questionNumber**. We'll use this to keep track of the question number that we're at.

```
struct ContentView: View {

    let questions = [Question(name: "Sample Question 1", answer: false),
Question(name: "Sample Question 2", answer: true)]
    @State var questionNumber = 0

    . . .
```

ContentView.swift

Showing the Question

- Next, let's change the text that we created earlier to show the current question.
  - To get the question that is in a particular index, we add it in the square brackets: **questions[questionNumber]**.
  - Then, to get the question name, we add a **.name** to it.

```
struct ContentView: View {

    let questions = [Question(name: "Sample Question 1", answer: false), Question(name:
"Sample Question 2", answer: true)]

    @State var questionNumber = 0

    var body: some View {
        VStack {
            Text(questions[questionNumber].name)

            . . .
```

ContentView.swift

Quiz Options

Time to make the buttons functional!

Clicky click click click click click click click click … click click click

If-statements!

- Using **if-statements**, we can run some code when a certain condition is met.
- Below is the syntax for **if-statements**.

```
if condition {
    // code to run if condition is met
} else {
    // code to run if condition is NOT met!
}
```

ContentView.swift

Conditions

- Conditions are ways to check things — for example, whether an integer is less than 5.
- In our case, we want to check whether the answer to the question is **true** or not.
- We can use the `==` operator for this. `==` will evaluate whether the left side of it is equals to the right side.

Is the answer correct?

- In the button for **true**, we can check whether the answer for that question is actually true. If the user got the answer correct, let's increase their score!

```
Button("True") {
    if questions[questionNumber].answer == true {
       score = score + 1
    }
}
```

ContentView.swift

Other button

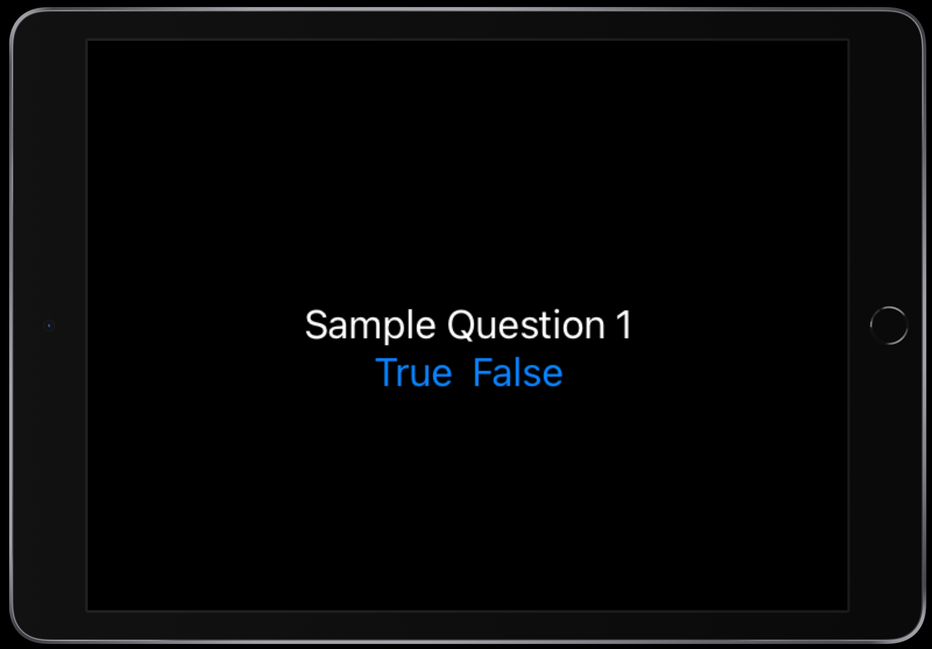
- Let's repeat the process for the **false** button, so the user can get a point if the correct answer is false.

```
Button("False") {
    if questions[questionNumber].answer == false {
        score = score + 1
    }
}
```

ContentView.swift

Checkpoint!

- At this point, your app should look like this.
- Check that you are able to proceed to the next question when you click 'True' or 'False'!
- Raise your hand if you need help.

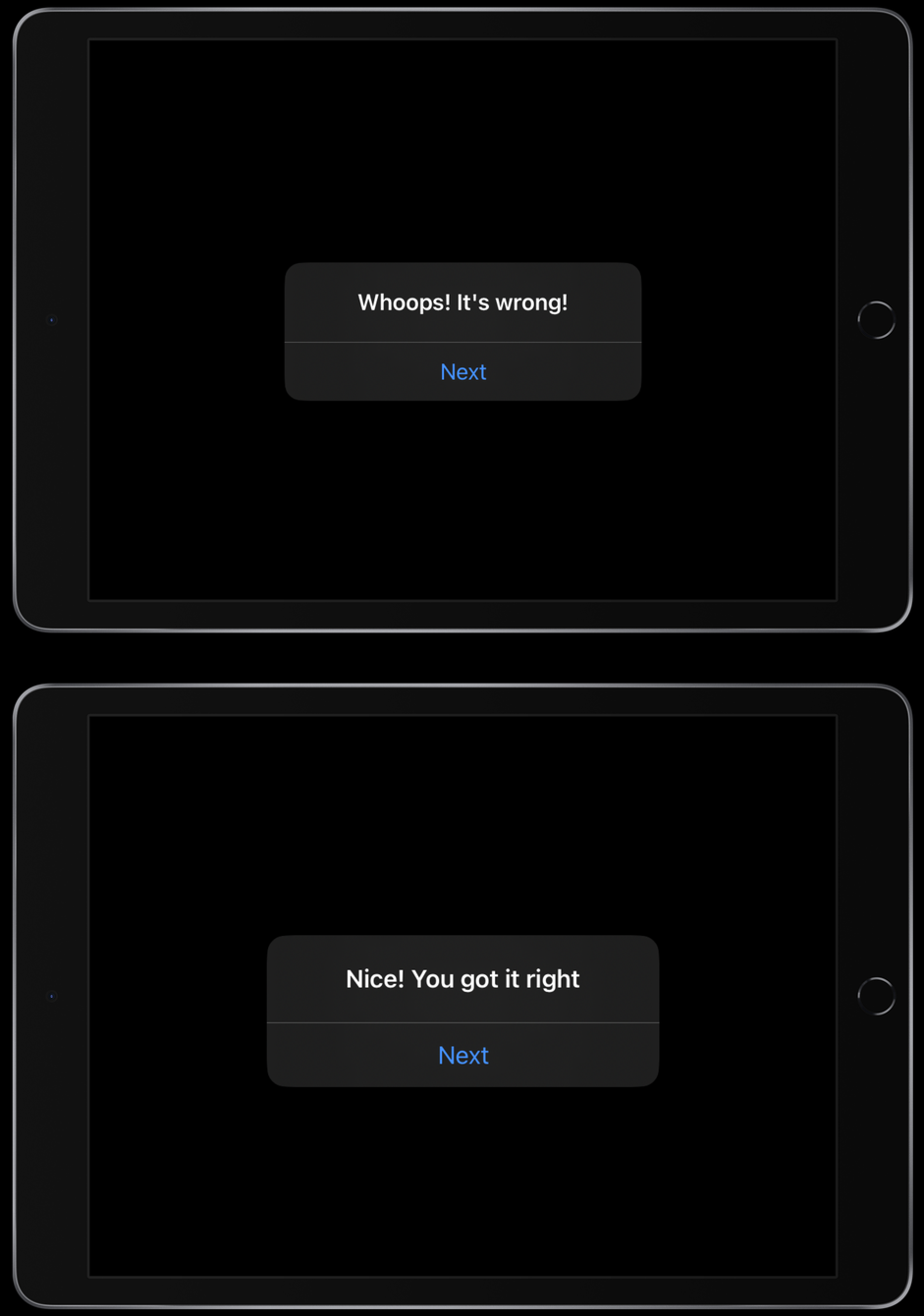


Presenting Alerts

Telling the user if they chose the correct or incorrect option

What is an Alert?

- It's a pop-up that asks the user for a response.
- You will find it in many apps to…
  - Ask the user about something, for example, permission to access the user's location
- In our app, we will create an alert like this 👉



Presenting Alerts

- To control the showing and hiding of an alert, we can make a @State variable called *presentAlert.*
- Also, to change the text shown in an alert, we can make another @State variable called *alertText.*

```
struct ContentView: View {

    let questions = [Question(name: "Sample Question 1", answer: false),
Question(name: "Sample Question 2", answer: true)]
    @State var questionNumber = 0
    @State var score = 0

    @State var presentAlert = false
    @State var alertText = ""

    . . .
```

ContentView.swift

Alerts

```
.alert(AlertTitle, isPresented: $presentAlert) {
    // The alert's actions
}
```

- This is just one way to create an alert in SwiftUI, but we will be using it as it is the simplest to understand.
  - `AlertTitle` is the place where the title of the alert should go.
  - `isPresented: $…` is where you should place the variable that controls when the alert is shown.

Creating Alerts in SwiftUI

Now, let's implement it into our code. We should also create a button to dismiss it.

For alerts, create it just **below the original VStack**.

```
.alert(alertText, isPresented: $presentAlert) {
    Button("Next") {

    }
}
```

ContentView.swift

Creating Alerts in SwiftUI

We should make it so that, when the ‘Next’ button is pressed, we should increase the value of questionNumber by 1 to go to the next question.

```
.alert(alertText, isPresented: $presentAlert) {
    Button("Next") {
        questionNumber = questionNumber + 1
    }
}
```

ContentView.swift

Showing the Alert

- When the user clicks each option button, we should present the alert to tell them if they are correct or not
  - So, let's change presentAlert to true and add a suitable alertText

```
Button("True") {
    presentAlert = true
    if questions[questionNumber].answer == true {
        score = score + 1
        alertText = "Nice! You got it right"
    }
}

Button("False") {
    presentAlert = true
    if questions[questionNumber].answer == false {
        score = score + 1
        alertText = "Nice! You got it right"
    }
}
```

ContentView.swift

Setting Alert Text

- In the **else** block, we can set the alert text to tell the user that they chose the wrong option :(
- Remember to do this for the **false** button too!

```
Button("True") {
    presentAlert = true
    if questions[questionNumber].answer == true {
        score = score + 1
        alertText = "Nice! You got it right"
    } else {
        alertText = "Whoops! It's wrong!"
    }
}
```

ContentView.swift

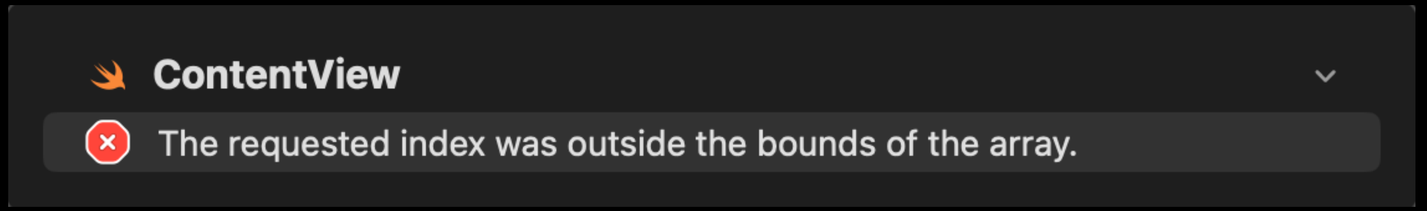
Run your code!

- When you run your code, you will be able to play with your trivia app!
- Wait… it crashes??
  - Why does it crash??????
  - Oh no!

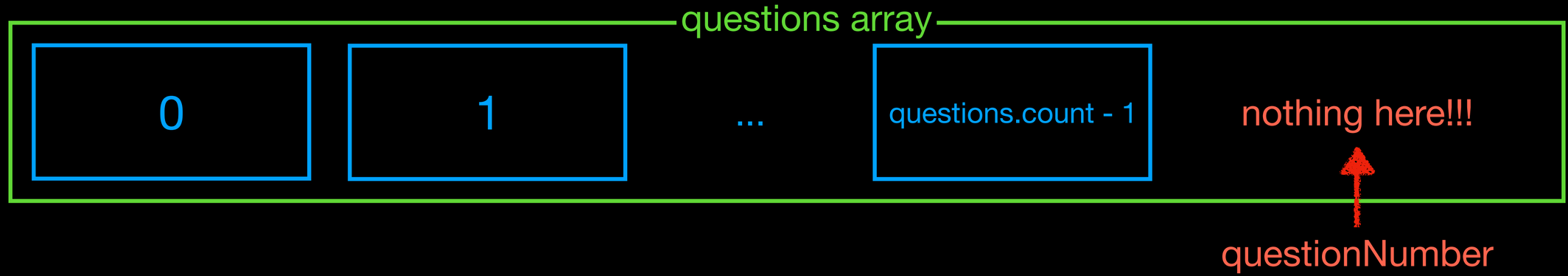
It CRASHEDD! :(

why crash?? i sad now

What error?



- WHAAAAAT?????
- Your app cannot continue, because you are trying to show a question that doesn't exist.
- Your index (`questionNumber`) has gone beyond the end of the current array (`questions.count`), even though the maximum index is (`questions.count - 1`).



How to fix?

- We need to add an if statement to check if the current index is equal to the number of questions.
  - If it is, we stop incrementing the index, else, we continue as usual. This avoids the code trying to find a question that does not exist.

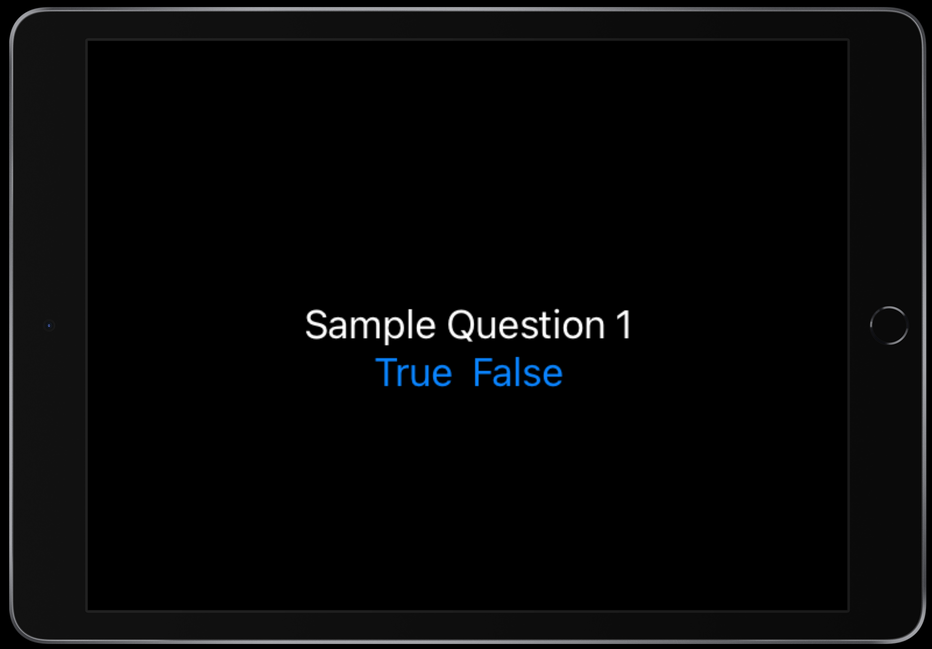
```
.alert(alertText, isPresented: $presentAlert) {
    Button("Next") {
        if questions.count == questionNumber + 1 {

        } else {
            questionNumber = questionNumber + 1
        }
    }
}
```

ContentView.swift

Checkpoint!

- At this point, your app is basically done!
- Ensure that you can proceed through all of the questions, and the answers are all correct.
- Raise your hand if you need help!



Presenting the Score

Showing the user their final score, and ~~whether they are a failure or not~~

Making it appear

- If you remember, we need to make a @State variable to control if the alert is shown.
- Create a new @State variable called *presentScore*

```
struct ContentView: View {

    let questions = [Question(name: "Sample Question 1", answer: false), Question(name:
"Sample Question 2", answer: true)]
    @State var questionNumber = 0
    @State var score = 0

    @State var presentAlert = false
    @State var alertText = ""

    @State var presentScore = false

    . . .
```

ContentView.swift

Making it appear

- When the question number is equals to the total number of questions, we present the score to the user

```
.alert(alertText, isPresented: $presentAlert) {
    Button("Next") {
        if questions.count == questionNumber + 1 {
            presentScore = true
        } else {
            questionNumber = questionNumber + 1
        }
    }
}
```

ContentView.swift

String Interpolation

- **String Interpolation** is the process of substituting values of certain variables into a string
  - For example, in our app, we need to use String Interpolation to show the score of the user
  - Why?
    - The score cannot be a fixed value, and has to be determined by the value of the *score* variable
- In Swift, String Interpolation can be done by typing \() in a string
  - The variable you want to add should be placed in between the parenthesis

Creating the Alert

You can copy over the code that you have wrote just now for the other Alert, but we have to change some things.

String Interpolation!

```
.alert("You got \(score) / \(questions.count)", isPresented: $presentScore){
    Button("Nice!") {

    }
}
```

ContentView.swift

Creating the Alert

When the user reaches the end, this alert is presented. When it is dismissed, we want to reset the score and question index so that the user can reattempt the quiz.

```
.alert("You got \(score) / \(questions.count)", isPresented: $presentScore){
    Button("Nice!") {
        score = 0
        questionNumber = 0
    }
}
```

ContentView.swift

(Optional) SwiftUI Modifiers

You can use Modifiers in SwiftUI to style your User Interface!

Modifiers

Image Modifiers

- `.cornerRadius()`
- `.mask()`
- `.resizable()`
- `.scaledToFit()`
- `.frame(width: 10, height: 10)`
- `.background()`

Padding

`.padding()`

abc Text Modifiers

- `.bold()`
- `.italic()`
- `.font(.system(size: 30))`
- `.font(.largeTitle)`
- `.foregroundColor()`
- `.background()`

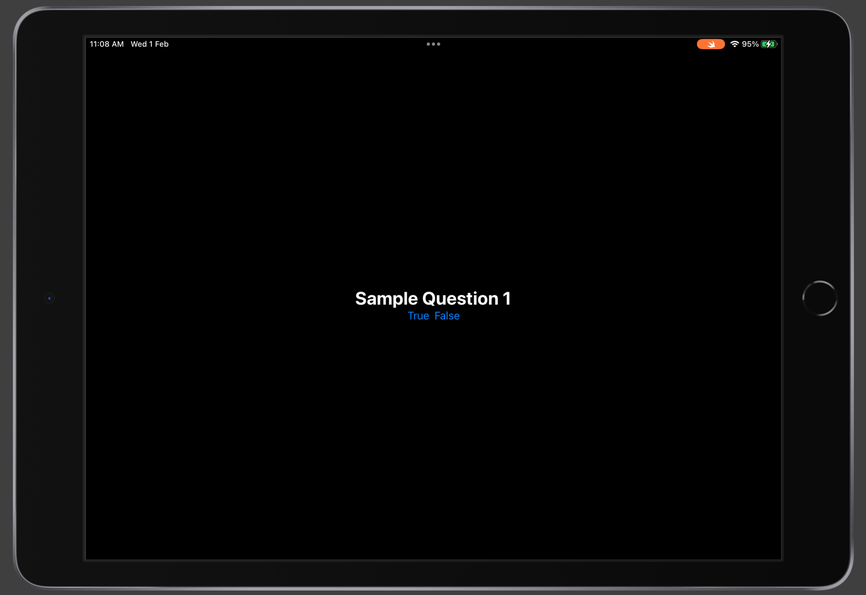
Text Modifiers

Now, the question title has the same font size as the button
This is to visually separate the title from the button

```
Text(questions[questionNumber].name)
    .font(.title)
    .bold()
```

ContentView.swift

Preview



Styling Buttons

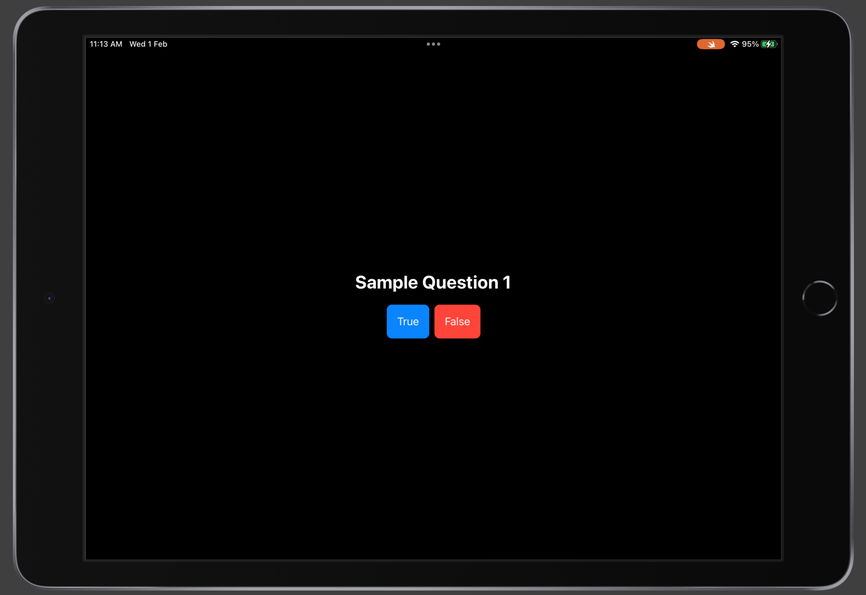
To make the buttons look nicer, we can give them a background! For example, the button for ‘true’ can be blue, while ‘false’ can be red

```
Button("True") {
    . . .
}
.padding()
.foregroundColor(.white)
.background(.blue)
.cornerRadius(8)

Button("False") {
    . . .
}
.padding()
.foregroundColor(.white)
.background(.red)
.cornerRadius(8)
```

ContentView.swift

Preview



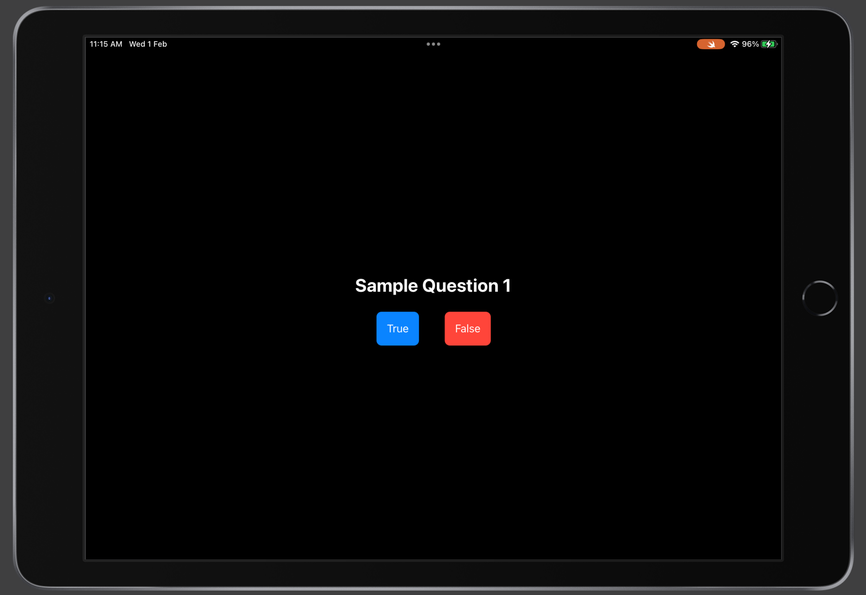
Adding some s p a c e

Nice! But, our User Interface feels very cramped now, let’s add some space!

```
VStack {
    Text(questions[questionNumber].name)
        .font(.title)
        .bold()
        .padding()
        HStack(spacing: 40) {
            . . .
```

ContentView.swift

Preview



```swift
import SwiftUI

struct ContentView: View {

    let questions = [Question(name: "Sample Question 1", answer: false), Question(name: "Sample Question 2", answer: true)]
    @State var questionNumber = 0
    @State var score = 0

    @State var presentAlert = false
    @State var alertText = ""

    @State var presentScore = false

    var body: some View {
        VStack {
            Text(questions[questionNumber].name)
                .font(.title)
                .bold()
                .padding()
            HStack(spacing: 40) {
                Button("True") {
                    presentAlert = true

                    if questions[questionNumber].answer == true {
                        score = score + 1
                        alertText = "Nice! You got it right"
                    } else {
                        alertText = "Whoops! It's wrong!"
                    }
                }
                .padding()
                .foregroundColor(.white)
                .background(.blue)
                .cornerRadius(8)

                Button("False") {
                    presentAlert = true

                    if questions[questionNumber].answer == false {
                        score = score + 1
                        alertText = "Nice! You got it right"
                    } else {
                        alertText = "Whoops! It's wrong!"
                    }
                }
                .padding()
                .foregroundColor(.white)
                .background(.red)
                .cornerRadius(8)
            }
        }
        .alert(alertText, isPresented: $presentAlert) {
            Button("Next") {
                if questions.count == questionNumber + 1 {
                    presentScore = true
                } else {
                    questionNumber = questionNumber + 1
                }
            }
        }
        .alert("You got \(score) / \(questions.count)", isPresented: $presentScore){
            Button("Nice!") {
                score = 0
                questionNumber = 0
            }
        }
    }
}
```

ContentView.swift

Any Questions?

We hope you learnt something, and most importantly… had fun!

hello@swiftinsg.org

swiftinsg.org

@swiftinsg