

2.3

Build a QuestionBot App

Build a QuestionBot App

What You'll Build

- The brains of an app bot that responds differently to different questions.

What You'll Learn

- How to take an idea you've coded in a playground and put it to use in a real app.
- How to make your app do different things in different situations.

Key Vocabulary

- **Case**
- **Default**

Introduction

In [Build a PhotoFrame App](#), you created an Xcode project from scratch. In this lesson, you'll add important functionality to an app that's already in progress.


Many apps are built by teams of people working together: designers who think about the way the app looks and how it's used, and developers who write the code to make it happen. Often, different parts of an app can be worked on at the same time.

In this lesson, you're part of a team that's building a chat app called QuestionBot. Focusing on only one part of the app, your job is to work on QuestionBot's "brain," the part that decides how to answer questions. Other parts of the app—the design, the user interface, the parts that take the question and display the answer—are already completed.

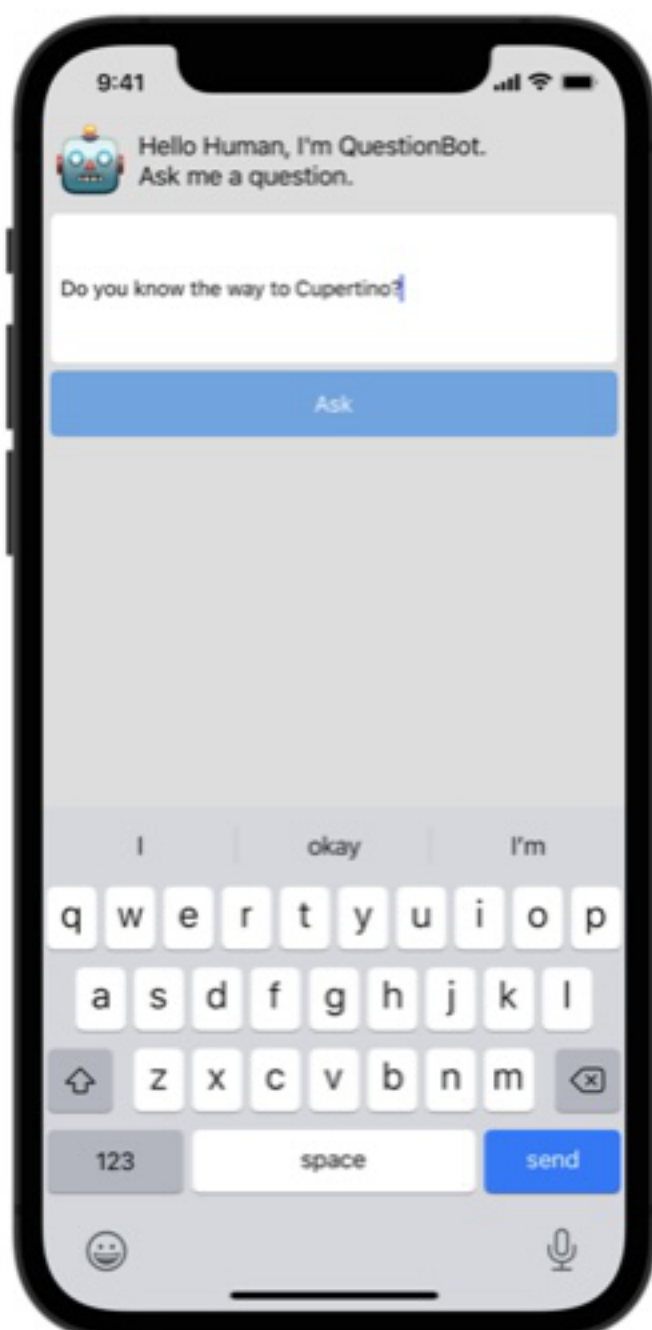
Right now, the app doesn't answer questions in any meaningful way. You'll build knowledge for the bot, and a personality to go with it—you'll give the app a unique behavior of your own design.

Part 1

Exploring The QuestionBot App

Open the project called **QuestionBot.xcodeproj** from the "QuestionBot" folder in your course resources and click the Run button  in the Xcode toolbar to run it.

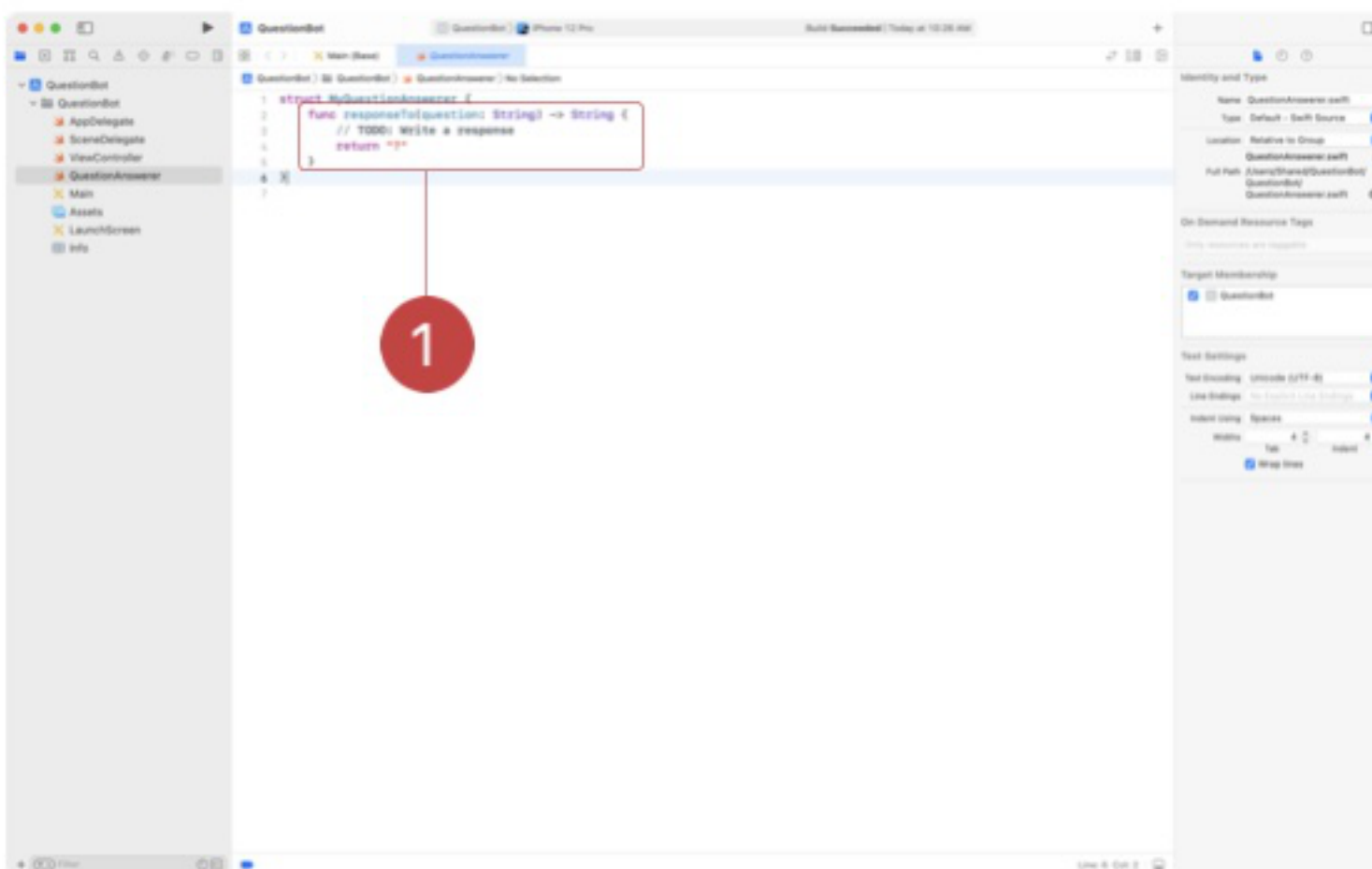
You'll see that whatever question you ask, you get the same answer — a question mark. It's your job to make QuestionBot give more useful answers, or at least fun answers.



Where Does Your Code Fit?

Looking in the **project navigator** (from the Xcode menu bar, choose **View > Navigators > Show Project Navigator**), there are only three code files in the app: **AppDelegate**, **ViewController**, and **QuestionAnswerer**. You could look into each file and try to figure out how the app works. And in the real world, sometimes that's the only information available to you.

In this case, the files have helpful names. Questions are answered in the **QuestionAnswerer**. file. Open the file and review the function highlighted in the image below: ¹



Here's the function. Notice a helpful `//TODO:` ("to do") **comment** tells you where you need to do your work. These two lines are the **code segment** you'll be replacing with your own code.

```
func responseTo(question: String) -> String {  
    // TODO: Write a response  
    return "?"  
}
```

The `responseTo(question:)` function takes a `String` and returns a `String`. Currently, the function ignores whatever question is passed and returns a question mark.

Changing the Answer

Change the function so it returns some text instead of a question mark.

1. Replace this line:

```
return "?"
```

With this line:

```
return "I'm sorry, I don't understand the  
question"
```

2. Build and run the app.
3. Ask a question. Your new answer is there!

This cycle of making a change, building and running, and testing the change is common. Sometimes, though, the part of the app you're working on requires you to get past a long series of screens, or it takes a long time to do whatever it is you're trying to test.



Remember

Remember, the function you're working on simply takes a string and gives a string back. It's independent from the rest of the app—in fact, you can build and test it without running the app at all.

This is a good way to build an app—when each working part knows as little as possible about the others, and each part has only one job to do. It means you can build each part separately, make sure it works exactly right, and then put them all together.

Rather than running the app several times as you develop your code, you can use a playground to get your question answerer up and running more quickly. You can see multiple results instantly, and there's no need to rerun. Once you're happy with your code, you'll use it to replace the existing code segment in QuestionBot—and you're done!

Part 2

Building Your Function

Open **QuestionAnswerer.playground** in your course resources and work through the exercises to build a working brain for QuestionBot.

Part 3

Updating the App

Putting It All Together

Once you've built the `responseTo(question:)` function in the playground, it's time to move it back to QuestionBot. You can do this by copying the code from the playground and pasting it into the app. It's important to be careful when doing this. Pasting the wrong part of code or pasting it in the wrong place can cause errors in the app project.



If you double-click the opening **brace** (`{`) of the function **body**, Xcode will automatically select everything up to the matching closing brace. Watch this video to see it in action.

1. On the Putting It All Together page of the playground, where you built the final version of the function, use the trick you just learned to select the entire function body, then copy the text using the Edit menu (**Edit > Copy**) or press Command-C (**⌘-C**).
2. Back in the QuestionBot project, open **QuestionAnswerer**. Select the entire body of the `responseToQuestion` function, then paste the code from the playground over the version in the app using the Edit menu (**Edit > Paste**) or press Command-V (**⌘-V**).
3. Build and run, then enjoy your new and improved QuestionBot.

Troubleshooting

If your app won't build and run, the problem likely has to do with pasting the code. Undo the paste you just did using the Edit menu (**Edit > Undo Paste**) or press **Command-Z** (⌘-Z), double-check the selected text and the location, and try again. For hints, look at how your app runs and check the errors that Xcode returns to you.

Following are some common problems:

- The app runs but you don't see the answers you were expecting. This is probably because you pasted the function in the wrong place. Perhaps it was in the wrong file, or in the wrong place in the correct file.
- The app doesn't build with an error that reads `Unexpected non-nil return value in void function`. You probably pasted the function body code from the playground into the wrong function in the project.
- The app doesn't build with an error that reads `Invalid redeclaration of responseTo(question:)`. This means you have two functions with the same name in the same area of the code—not allowed because it isn't clear which function should run when called.
- The app doesn't build with an error that reads `Expected declaration`. You may have inadvertently replaced the entire function, declaration and all, with only the body of the function from your playground, or you pasted the function body into the wrong place.
- The app doesn't build with an error that reads `Missing return in a function expected to return 'String'`. You probably pasted an entire function inside another one.

- The app doesn't build with an error that reads `Closure expression is unused` or `Expressions are not allowed at the top level`. You probably pasted the body of your function into a file without any context—with no `func` keyword.
- The app doesn't build with an error that reads `Use of unresolved identifier 'MyQuestionAnswerer'`. You probably replaced the entire `struct MyQuestionAnswerer` with the `responseToQuestion` function or its body.

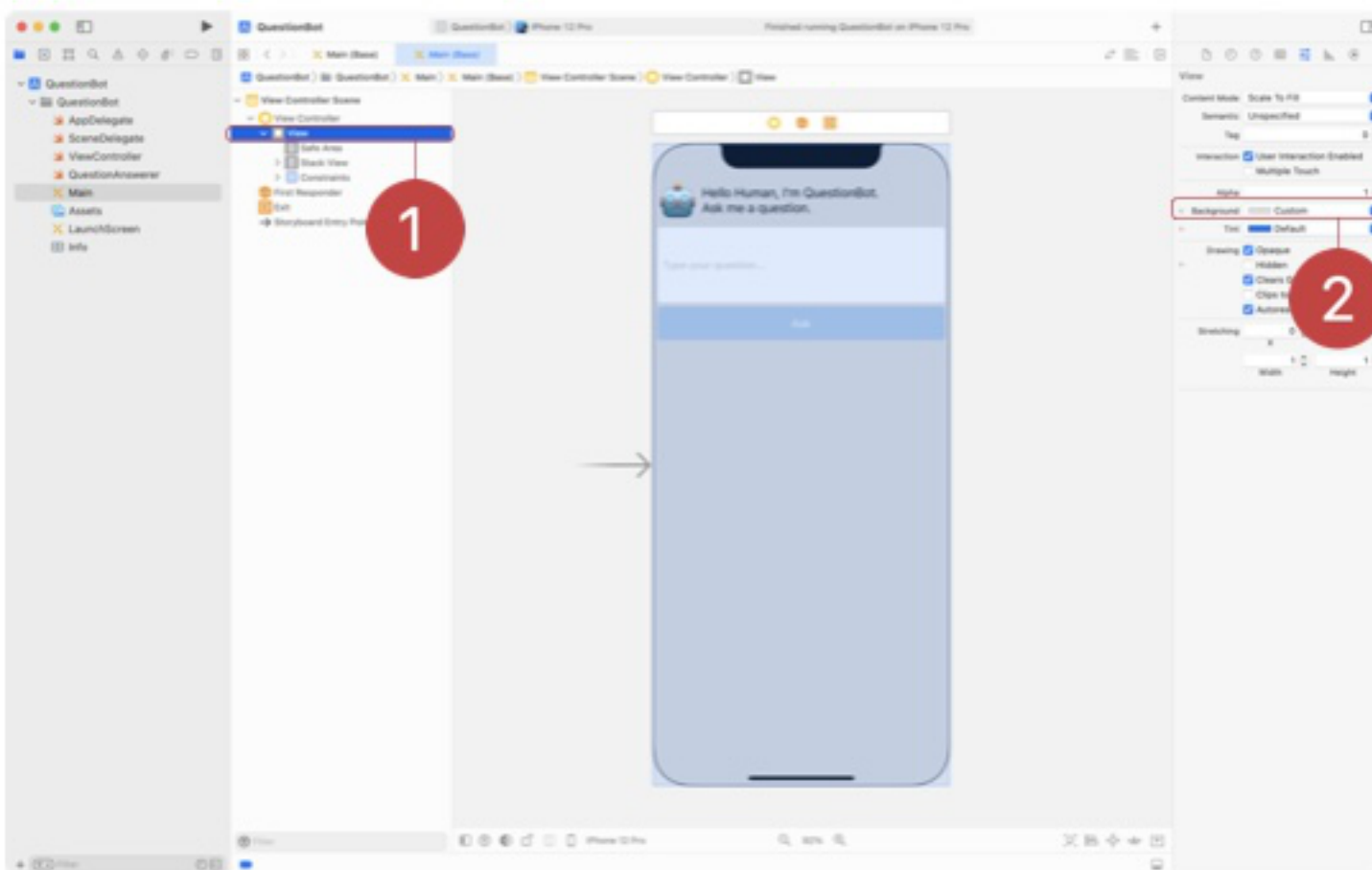
Part 4 (Optional)

Customizing the Interface

You can use Interface Builder to change the way QuestionBot looks. Open the Main storyboard, and make sure the Attributes inspector is visible. See [Build a PhotoFrame App](#) for a refresher on how to do this.

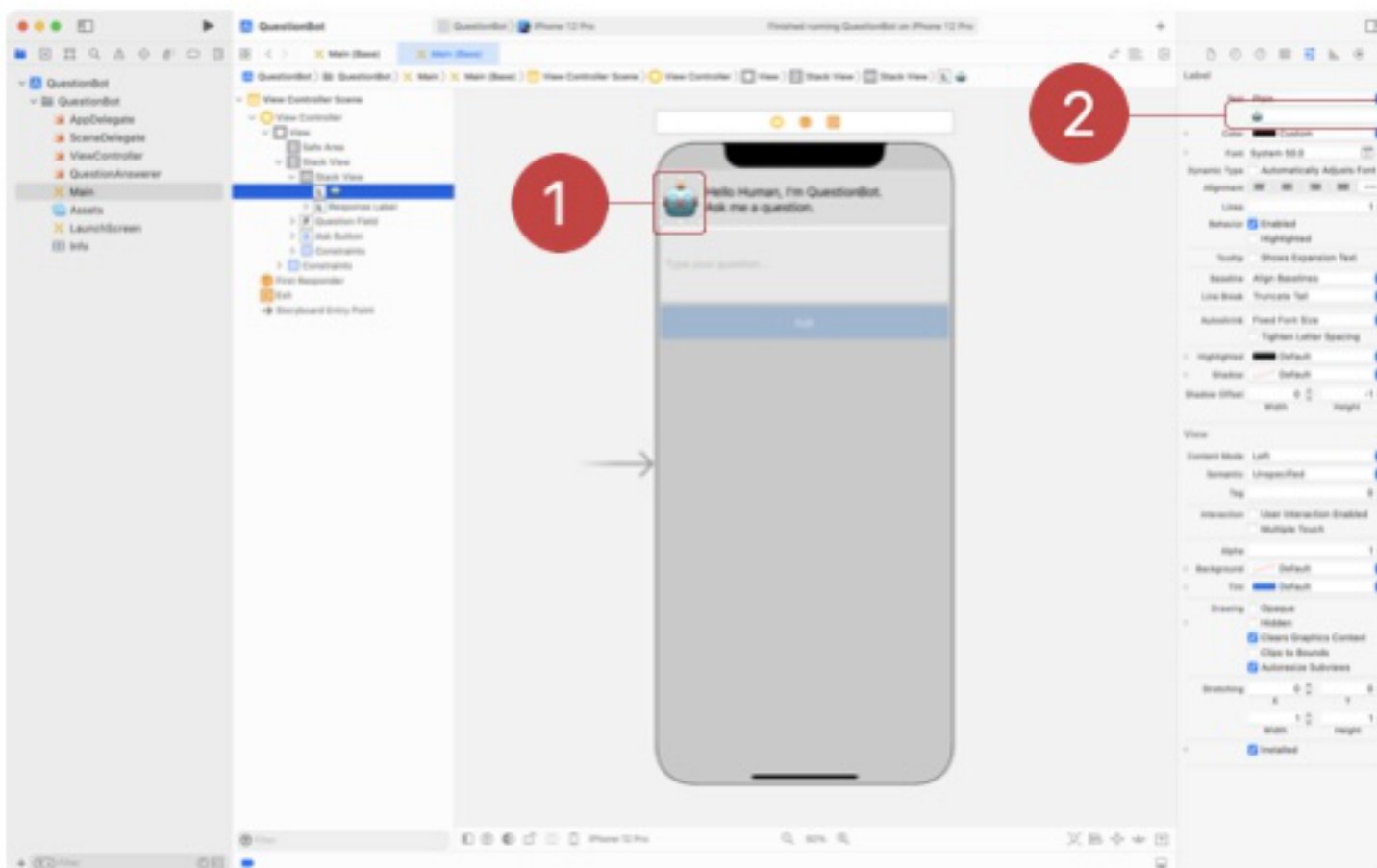
Background Color

Select View¹, as shown in the Document Outline below. Use the Background² attribute to choose a different color.



Robot Head

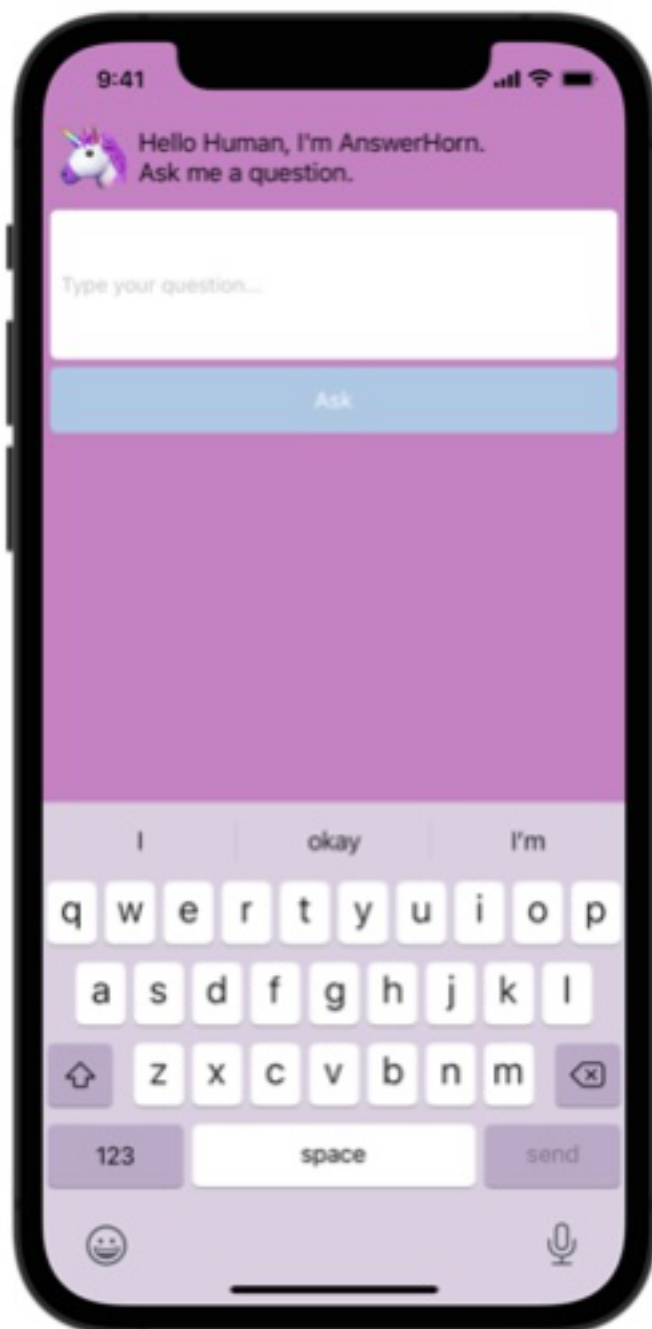
Select the 🤖 label ¹ as shown in the Document Outline below. Using the Text attribute ² in the inspector, change the emoji to a different character. Press **Ctrl-⌘-Space** to bring up the emoji picker.



Welcome Message

Select Response Label in the Document Outline and change the opening text, pressing **Ctrl-Return** to add a new line if you need one.

Build and run to see your changes, which should look something like the screenshot below.



Reflection Questions

Many apps use the same architecture as QuestionBot, where the app has a separate “brain” that tells the user interface what to display.

For example, QuestionBot takes in a string question and returns a string answer. So if you input “Should I sing out loud?”, the result will be “Probably.” Think of an app that you use, and imagine what its brain’s job would be.

For the app you’re thinking of, what’s an example of an input the brain would take?

What’s an example of a result it would return to display in the user interface?

Summary

You saw how to use a playground to adjust one piece of logic or interaction, then bring the refined code back into the app once it was just right. By focusing on one piece of a project at a time, you'll find it's easier to be sure each piece works well by itself. And when you build those well-functioning individual pieces into a program, you'll have a much more powerful app.