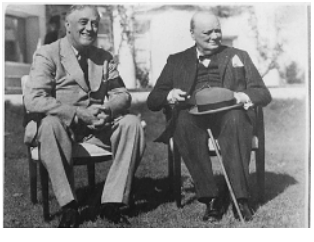
CHAPTER 8

Presidents Quiz



The Presidents Quiz is a trivia game about former leaders of the United States. Though this quiz is about presidents, you can use it as a template to build quizzes or study guides on any topic.

In the previous chapters, you've been introduced to some fundamental programming concepts. Now, you're ready for something more challenging. You'll find that this chapter requires a conceptual leap in terms of programming skills and abstract thinking. In particular, you'll use two list variables to store the data—in this case, the quiz questions and answers—and you'll use an index variable to track where the user is in the quiz. When you finish, you'll be armed with the knowledge to create quiz apps and many other apps that require list processing.

This chapter assumes that you're familiar with the basics of App Inventor: using the Component Designer to build the user interface, and using the Blocks Editor to specify event handlers and program the component behavior. If you are not familiar with these fundamentals, be sure to review the previous chapters before continuing.

You'll design the quiz so that the user proceeds from question to question by clicking a Next button and receives feedback about the answers.

What You'll Learn

This app, shown in Figure 8-1, covers:

- Defining list variables for storing the questions and answers in lists.
- Sequencing through a list using an index; each time the user clicks Next, you'll display the next question.
- Using conditional (if) behaviors: performing certain operations only under specific conditions. You'll use an `if` block to handle the app's behavior when the user reaches the end of the quiz.

Figure 8-1. The Presidents Quiz in action

- Switching an image to show a different picture for each quiz question.

Getting Started

Navigate to the App Inventor website and start a new project. Give the project the name "PresidentsQuiz" and set the screen's title to "Presidents Quiz". Connect your device or emulator for live testing.

You'll need to download the following pictures for the quiz from the *appinventor.org* website onto your computer:

- *http://appinventor.org/bookFiles/PresidentsQuiz/roosChurch.gif*
- *http://appinventor.org/bookFiles/PresidentsQuiz/nixon.gif*
- *http://appinventor.org/bookFiles/PresidentsQuiz/carterChina.gif*
- *http://appinventor.org/bookFiles/PresidentsQuiz/atomic.gif*

You'll load these images into your project in the next section.

Designing the Components

The Presidents Quiz app has a simple interface for displaying the questions and allowing the user to answer. You can build the components from the snapshot of the Component Designer shown in Figure 8-2.

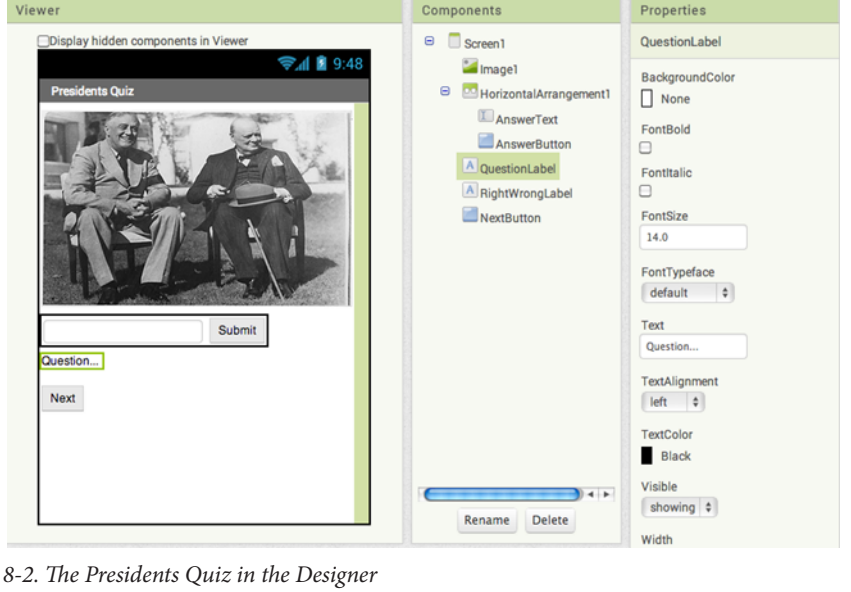


Figure 8-2. The Presidents Quiz in the Designer

To create this interface, first load the images you downloaded into the project. In the Media area, click Add and select one of the downloaded files (e.g., *roosChurch.gif*). Do the same for the other three images. Then, add the components listed in Table 8-1.

Table 8-1. Components for the Presidents Quiz app

Component type	Palette group	What you'll name it	Purpose
`Image`	User Interface	`Image1`	The picture displayed with the question.
`Label`	User Interface	`QuestionLabel`	Display the current question.
`HorizontalArrangement`	Layout	`HorizontalArrangement1`	Put the answer text and button in a row.
`TextBox`	User Interface	`AnswerText`	The user will enter his answer here.
`Button`	User Interface	`AnswerButton`	The user clicks this to submit an answer.
`Label`	User Interface	`RightWrongLabel`	Display "correct!" or "incorrect!"
`Button`	User Interface	`NextButton`	The user clicks this to proceed to the next question.

1. Set `Image1.Picture` to the image file *roosChurch.gif*, the first picture that should appear. Set its Width to "Fill parent" and its Height to 200.
2. Set `QuestionLabel.Text` to "Question..." (you'll input the first question in the Blocks Editor).
3. Set `AnswerText.Hint` to "Enter an answer". Set its `Text` property to blank. Move it into `HorizontalArrangement1`.
4. Change `AnswerButton.Text` to "Submit" and move it into `HorizontalArrangement1`.
5. Change `NextButton.Text` to "Next".
6. Change `RightWrongLabel.Text` to blank.

Adding Behaviors to the Components

You'll need to program the following behaviors:

- When the app starts, the first question appears, including its corresponding image.
- When the user clicks the `NextButton`, the second question appears. When it's clicked again, the third question appears, and so on.

- When the user reaches the last question and clicks the `NextButton`, the first question should appear again.
- When the user answers a question, the app will report whether it is correct.

You'll code these behaviors one by one, testing as you go.

Defining the Question and Answer Lists

To begin, define two list variables based on the items in Table 8-2: `QuestionList` to hold the list of questions, and `AnswerList` to hold the list of corresponding answers. Figure 8-3 shows the two lists you'll create in the Blocks Editor.

Table 8-2. Variables for holding question and answer lists

Block type	Drawer	Purpose
`initialize global to` ("QuestionList")	Variables	Store the list of questions (rename it `QuestionList`).
`initialize global to`("AnswerList")	Variables	Store the list of answers (rename it `AnswerList`).
`make a list`	Lists	Insert the items of the `QuestionList`.
`text` (three of them)	Text	The questions.
`make a list`	Lists	Insert the items of the `AnswerList`.
`text` (three of them)	Text	The answers.

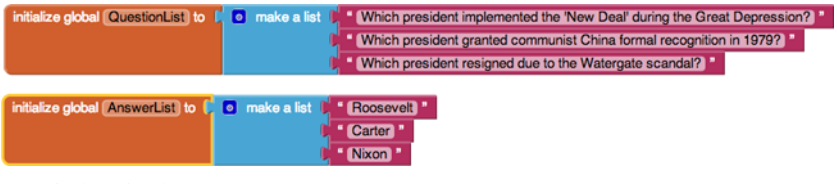


Figure 8-3. The lists for the quiz

Defining the Index Variable

The app needs to keep track of the current question as the user clicks the `NextButton` to proceed through the quiz. You'll define a variable named `currentQuestionIndex` for this, and the variable will serve as the index into both the `QuestionList` and `AnswerList`. Table 8-3 lists the blocks you'll need to do this, and Figure 8-4 shows what that variable will look like.

Table 8-3. Creating the index

Block type	Drawer	Purpose
`initialize global to` ("currentQuestionIndex")	Variables	Hold the index (position) of the current question/answer.
`number` (1)	Math	Set the initial value of `currentQuestionIndex` to 1 (the first question).

Figure 8-4. Initiating the index variable with a value of 1

Displaying the First Question

Now that you've defined the variables you need, you can specify the app's interactive behavior. As with any app, it's important to work incrementally and define one behavior at a time. To begin, think only about the questions and, specifically, displaying the first question on the list when the app launches. We'll come back and deal with the images and answers a bit later.

You want your code blocks to work regardless of the specific questions that are on the list. That way, if you decide to change the questions or create a new quiz by copying and modifying this app, you'll only need to change the actual questions in the list definitions, and you won't need to change any event handlers.

So, for this first behavior, you don't want to refer directly to the first question, "Which president implemented the 'New Deal' during the Great Depression?" Instead, you want to refer, abstractly, to the first socket in the `QuestionList` (regardless of the specific question there). That way, the blocks will still work even if you modify the question in that first socket.

You select particular items in a list with the `select list item` block. The block asks you to specify the list and an index (a position in the list). If a list has three items, you can enter 1, 2, or 3 as the index.

For this first behavior, when the app launches, you want to select the first item in `QuestionList` and place it in the `QuestionLabel`. Recall from the Android, Where's My Car? app, if you want something to happen when your app launches, you program that behavior in the `Screen1.Initialize` event handler. You can use the blocks listed in Table 8-4.

Table 8-4. Blocks to load the initial question when the app starts

Block type	Drawer	Purpose
`Screen1.Initialize`	Screen1	Event handler triggered when the app begins.
`set QuestionLabel.Text to`	QuestionLabel	Put the first question in `QuestionLabel`.
`select list item`	Lists	Select the first question from `QuestionList`.
`get global QuestionList`	Variables	The list to select questions from.
`number` (1)	Math	Select the first question by using an index of 1.

How the blocks work

The `Screen1.Initialize` event is triggered when the app starts. As shown in Figure 8-5, the first item of the variable `QuestionList` is selected and placed into `QuestionLabel.Text`. So, when the app launches, the user will see the first question.

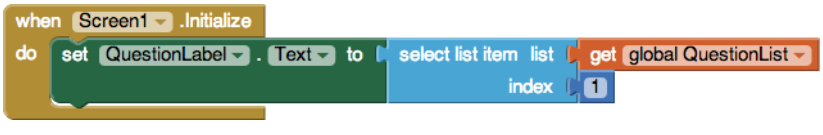


Figure 8-5. Selecting the first question

Test your app *Click Connect and connect to your device or the emulator for live testing. When your app loads, do you see the first item of* `QuestionList`*, "Which president implemented the 'New Deal' during the Great Depression?"*

Iterating Through the Questions

Now, program the behavior of the `NextButton`. You've already defined the `currentQuestionIndex` to remember the current question. When the user clicks the `NextButton`, the app needs to increment the `currentQuestionIndex` (i.e., change it from 1 to 2 or from 2 to 3, and so on). You'll then use the resulting value of `currentQuestionIndex` to select the new question to display. As a challenge, see if you can build these blocks on your own. When you're finished, compare your results against Figure 8-6.

Figure 8-6. Moving to the next question

How the blocks work

The first row of blocks increments the variable `currentQuestionIndex`. If `currentQuestionIndex` has a 1 in it, it is changed to 2. If it has a 2, it is changed to 3, and so on. After the `currentQuestionIndex` variable has been changed, the app uses it to select the new question to display. When the user clicks `NextButton` for the first time, the increment blocks will change `currentQuestionIndex` from 1 to 2, so the app will select the second item from `QuestionList`, "Which president granted communist China formal recognition in 1979?" The second time `NextButton` is clicked, `currentQuestionIndex` will be set from 2 to 3, and the app will select the third question on the list, "Which president resigned due to the Watergate scandal?"

Note *Take a minute to compare the blocks of* `NextButton.Click` *to those in the* `Screen.Initialize` *event handler. In the* `Screen.Initialize` *blocks, the app used* `select list item` *with a concrete number (1) to select the list item. In these blocks, you're selecting the list item by using a variable as the index. The app doesn't choose the first item in the list, or the second, or the third; it chooses the* `currentQuestionIndexth` *item, and thus a different item will be selected each time the user clicks* `NextButton`*. This is a very common use for an index—incrementing its value to find and display or process items in a list.*

Test your app *Test the behavior of the* `NextButton` *to see if the app is working correctly. Click the* `NextButton` *on the phone. Does the phone display the second question, "Which president granted communist China formal recognition in 1979?" It should, and the third question should appear when you click the* `NextButton` *again. But if you click again, you should see an error: "Attempting to get item 4 of a list of length 3." The app has a bug! Do you know what the problem is? Try figuring it out before moving on.*

The problem with the code thus far is that it simply increments to the next question each time without any concern for the end of the quiz. When `currentQuestionIndex` is already 3 and the user clicks the`NextButton`, the app changes `currentQuestionIn dex` from 3 to 4. It then calls `select list item` to get the `currentQuestionIndexth` item—in this case, the fourth item. Because there are only three items in the variable `QuestionList`, the Android device doesn't know what to do. As a result, it displays an error and forces the app to quit. How can we let the app know that it has reached the end of the quiz?

The app needs to ask a question when the `NextButton` is clicked, and execute different blocks depending on the answer. Because you know your app contains three questions, one way to ask the question would be, "Is the variable `currentQuestionIn dex` greater than 3?" If the answer is yes, you should set `currentQuestionIndex` back to 1 and take the user back to the first question. The blocks you'll need for this are listed in Table 8-5.

Table 8-5. Blocks for checking the index value for the end of the list

Block type	Drawer	Purpose
`if`	Control	Figure out if the user is on the last question.
`>=`	Math	Test if `currentQuestionIndex` is greater than 3.
`get global current QuestionIndex`	Drag from initialize block	Put this into the left side of =.
`number 3`	Math	Put this into the right side of = because 3 is the number of items in the list.
`set global current QuestionIndex to`	Drag from initialize block	Set to 1 to revert to the first question.
`number 1`	Math	Set the index to 1.

The blocks should appear as illustrated in Figure 8-7.

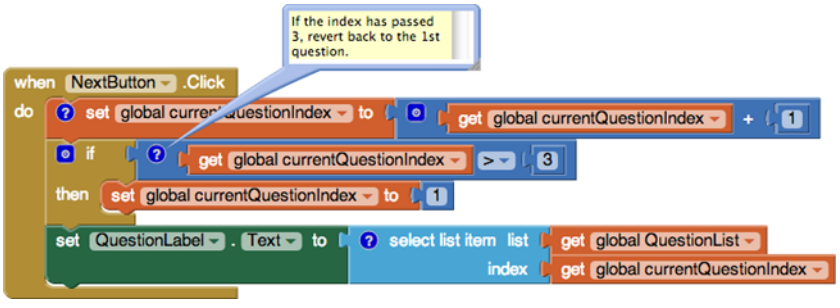


Figure 8-7. Checking if the index is past the last question

How the blocks work

When the user clicks the `NextButton`, the app increments the index as it did before. But then, as is shown in Figure 8-8, it checks to see if `currentQuestionIndex` is greater than 3, which is the number of questions. If it is greater than 3, `currentQuestionIndex` is set back to 1, and the first question is displayed. If it is 3 or less, the blocks within the `if` block are not performed, and the current question is displayed as usual.

Test your app *Pick up the phone and click the* `NextButton`*. The second question, "Which president granted communist China formal recognition in 1979?" should appear in the* `QuestionLabel` *on the phone, as before. When you click again, the third question should appear on the phone. Now, for the behavior you're really testing: if you click again, you should see the first question ("Which president implemented the 'New Deal' during the Great Depression?") appear on the phone.*

Making the Quiz Easy to Modify

If your blocks for the `NextButton` work, pat yourself on the back; you are on your way to becoming a programmer! But what if you added a new question (and answer) to the quiz? Would your blocks still work? To explore this, first add a fourth question to `QuestionList` and a fourth answer into `AnswerList`, as shown in Figure 8-8.

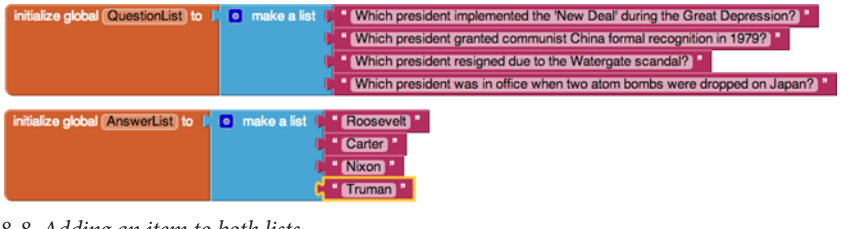


Figure 8-8. Adding an item to both lists

Test your app *Click the* `NextButton` *several times. You'll notice that the fourth question never appears, no matter how many times you click Next. Do you know what the problem is? Before reading on, see if you can fix the blocks so that the fourth question appears.*

The issue here is that the test to determine whether the user is on the last question is too specific; it asks if the `currentQuestionIndex` variable is greater than 3. You could just change the number 3 to a 4, and the app would work correctly again. The problem with that solution, however, is that each time you modify the questions and answer lists, you must also remember to make this change to the if-test.

Such dependencies in a computer program often lead to bugs, especially as an app grows in complexity. A much better strategy is to design the blocks so that they will work no matter how many questions there are. Such generality makes it easier if you, as a programmer, want to customize your quiz for some other topic. It is also essential if the list you are working with changes dynamically—for example, think of a quiz app that allows the user to add new questions (you'll build this in Chapter 10). For a program to be more general, it can't refer to concrete numbers such as 3, because that only works for quizzes of three questions.

So, instead of asking if the value of `currentQuestionIndex` is greater than the specific number 3, ask if it is greater than the number of items in `QuestionList`. If the app asks this more general question, it will work even when you add to or remove items from the `QuestionList`. Let's modify the `NextButton.Click` event handler to replace the previous test that referred directly to 3. You'll need the blocks listed in Table 8-6.

Table 8-6. Blocks to check the length of the list

Block type	Drawer	Purpose
`length of list`	Lists	Ask how many items are in `QuestionList`.
`get global QuestionList`	Drag in from initialization block	Put this into the "list" socket of `length of list`.

How the blocks work

The `if` test now compares the `currentQuestionIndex` to the length of the `QuestionList`, as shown in Figure 8-11. So, if `currentQuestionIndex` is 5, and the length of the `QuestionList` is 4, then the `currentQuestionIndex` will be set back to 1. Note that, because the blocks no longer refer to 3 or any specific number, the behavior will work no matter how many items are in the list.

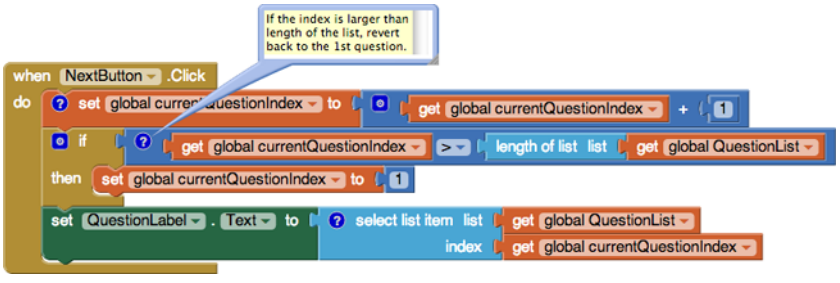


Figure 8-9. Checking if the index is larger than length of list (instead of 3)

Test your app When you click the `NextButton`, does the app now cycle through the four questions, moving to the first one after the fourth?

Switching the Image for Each Question

Now that you've programmed all the behaviors for moving through the questions (and you've made your code smarter and more flexible by making it more abstract), your next task is to get the images working properly, too. Right now, the app shows the same image no matter what question is being asked. You can change this so an image pertaining to each question appears when the user clicks the `NextButton`. Earlier, you added four pictures as media for the project. Now, you'll create a third list, `PictureList`, with the image filenames as its items. You'll also modify the `NextButton.Click` event handler to switch the picture each time, just as you switch the question text. (If you're already thinking about using the `currentQuestionIndex` here, you're on the right track!) First, create a `PictureList` and initialize it with the names of the image files. Be sure that the names are exactly the same as the filenames you loaded into the Media section of the project. Figure 8-10 shows how the blocks for the `PictureList` should look.

Figure 8-10. The PictureList with image filenames as items

Next, modify the `NextButton.Click` event handler so that it changes the picture that appears for each question. The `Image.Picture` property is used to change the picture displayed. To modify `NextButton.Click`, you'll need the blocks listed in Table 8-7.

Table 8-7. Blocks to add the image that accompanies the question

Block type	Drawer	Purpose
`set Image1.Picture to`	Image1	Set this to change the picture.
`select list item`	Lists	Select the picture corresponding to the current question.
`get global PictureList`	Drag out from initialization block	Select a filename from this list.
`get global currentQuestionIndex`	Drag out from initialization block	Select the `currentQuestionIndex` item.

How the blocks work

The `currentQuestionIndex` serves as the index for both the `QuestionList` and the `PictureList`. As long as you've set up your lists properly such that the first question corresponds to the first picture, the second to the second, and so on, the single index can serve both lists, as shown in Figure 8-11. For instance, the first picture, *roosChurch.gif*, is a picture of President Franklin Delano Roosevelt (sitting with British Prime Minister Winston Churchill), and "Roosevelt" is the answer to the first question.

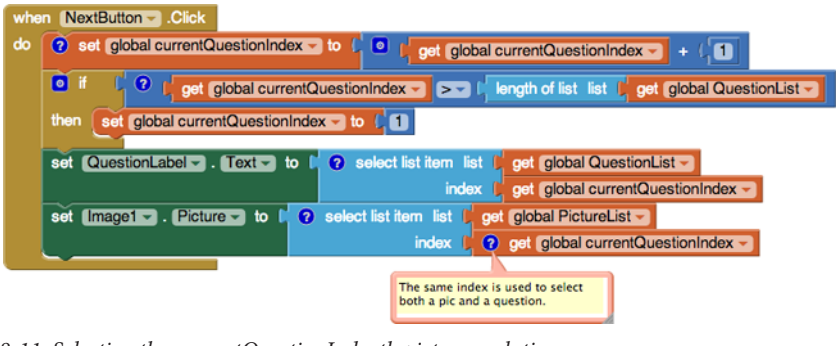


Figure 8-11. Selecting the currentQuestionIndexth picture each time

Test your app *Click Next a few times. Does a different image appear each time you click the* `NextButton`?

Checking the User's Answers

Thus far, you've created an app that simply cycles through questions and answers (paired with an image of the answer). It's a great example of apps that use lists, but to

be a true quiz app, it needs to give users feedback on whether their answers are correct. Let's add the blocks to do just that. Our interface is set up so that the user types an answer in `AnswerText` and then clicks the `AnswerButton`. The app must compare the user's entry with the answer to the current question, using an `if else` block to check. The `RightWrongLabel` should then be modified to report whether the answer is correct. There are quite a few blocks needed to program this behavior, all of which are listed in Table 8-8.

Table 8-8. Blocks for indicating whether an answer is correct

Block type	Drawer	Purpose
`AnswerButton.Click`	AnswerButton	Triggered when the user clicks the `AnswerButton`.
`ifelse`	Control	If the answer is correct, do one thing; otherwise, do another.
`=`	Math	Ask if the answer is correct.
`AnswerText.Text`	AnswerText	Contains the user's answer.
`select list item`	Lists	Select the current answer from `AnswerList`.
`get global AnswerList`	Drag out from initialization block	The list to select from.
`get global currentQuestionIndex`	Drag out from initialization block	The current question (and answer) number.
`set RightWrongLabel.Text to`	RightWrongLabel	Report the answer here.
`text` ("correct!")	Text	Display this if the answer is right.
`set RightWrongLabel.Text to`	RightWrongLabel	Report the answer here.
`text` ("incorrect!")	Text	Display this if the answer is wrong.

How the blocks work

Figure 8-12 demonstrates how the `if else` test asks whether the answer the user provided (`AnswerText.Text`) is equal to the `currentQuestionIndexth` item in the `AnswerList`. If `currentQuestionIndex` is 1, the app will compare the user's answer with the first item in `AnswerList`, "Roosevelt." If `currentQuestionIndex` is 2, the app will compare the user's answer with the second answer in the list, "Carter," and so on. If the test result is positive, the `then` branch is executed and the `RightWrongLabel` is set to "correct!" If the test is false, the `else` branch is executed and the `RightWrongLabel` is set to "incorrect!"

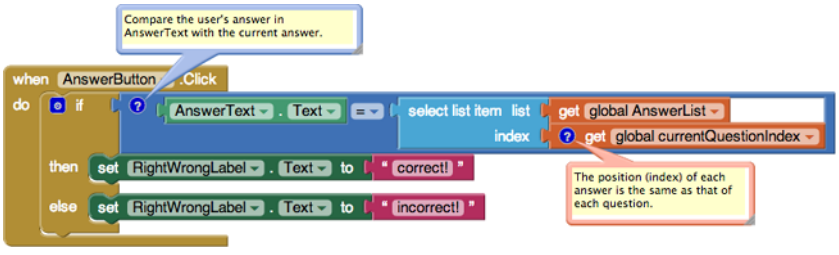


Figure 8-12. Checking the answer

Test your app *Try answering one of the questions. It should report whether you answered the question exactly as specified in the* `AnswerList`. *Test with both a correct and incorrect answer. You'll likely notice that for an answer to be marked as correct, it has to be an exact match and case-specific to what you entered in the* `AnswerList`. *Be sure to also test that things work on successive questions.*

The app should work, but you might notice that when you click the `NextButton`, the "correct!" or "incorrect!" text and the previous answer are still there, as shown in Figure 8-13, even though you're looking at the next question. It's fairly innocuous, but your app users will definitely notice such user interface issues. To blank out the `RightWrongLabel` and the `AnswerText`, you'll put the blocks listed in Table 8-9 within the `NextButton.Click` event handler.

Figure 8-13. The quiz in action with the previous answer appearing when it shouldn't

Table 8-9. Blocks to clear the RightWrongLabel

Block type	Drawer	Purpose
`set RightWrongLabel.Text to`	RightWrongLabel	This is the label to blank out.
`text` ("")	Text	When the user clicks `NextButton`, clear the previous answer's feedback.
`set AnswerText.Text to`	AnswerText	The user's answer from the previous question.
`text` ("")	Text	When the user clicks the `NextButton`, clear the previous answer.

How the blocks work

As shown in Figure 8-14, by clicking the `NextButton`, the user moves on to the next question, so the top two rows of the event handler clear out the `RightWrongLabel` and the `AnswerText`.

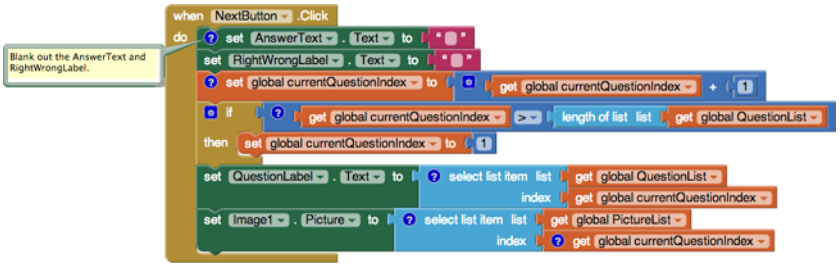


Figure 8-14. The PictureList with image filenames as items

Test your app *Answer a question and click Submit, then click the* `NextButton`. *Did your previous answer and its feedback disappear?*

The Complete App: The Presidents Quiz

Figure 8-15 shows the final block configuration for the Presidents Quiz.

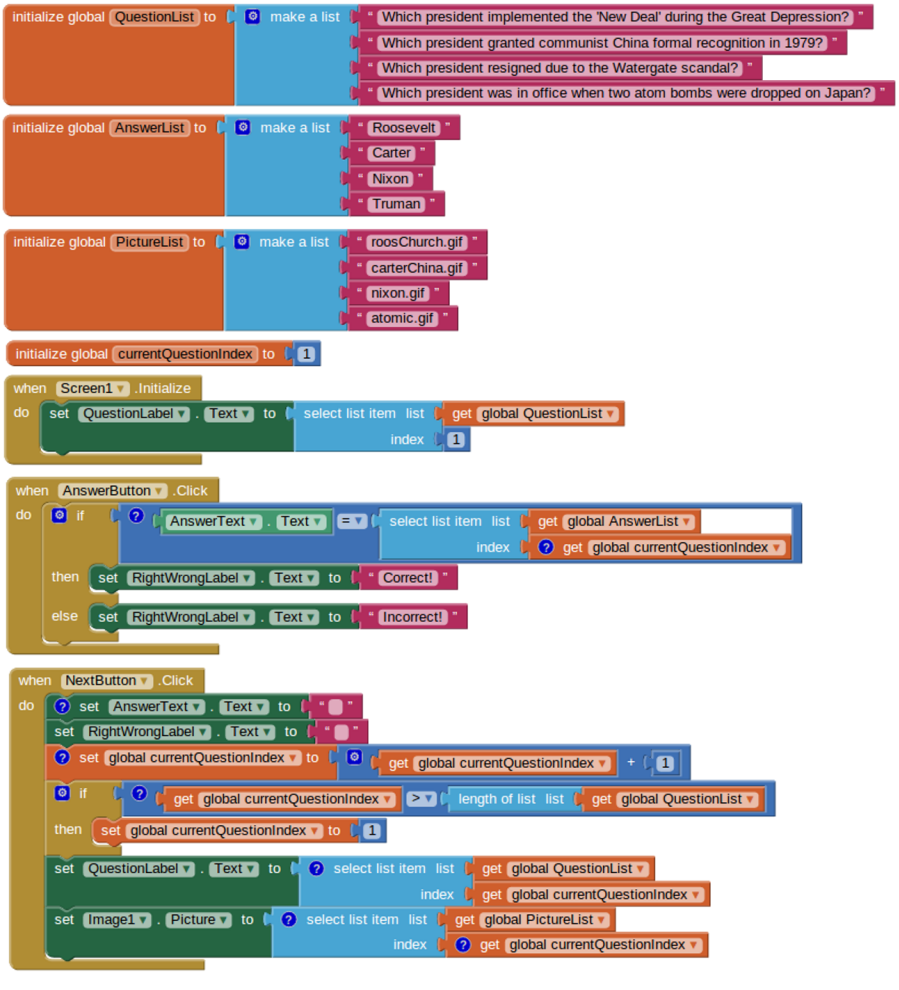


Figure 8-15. The blocks for the Presidents Quiz

Variations

When you get this quiz working, you might want to explore some variations, such as the following:

- Instead of just showing images for each question, try playing a sound clip or a short video. With sound, you can turn your quiz into a Name That Tune app.
- The quiz is very rigid in terms of what it accepts as a valid answer. There are a number of ways to improve this, taking advantage of text processing blocks in the text drawer. One way is to use the upcase block in the Text drawer to convert the user's answer and actual answer to all upper case before comparing. Another

is to use the `text.contains` block to see if the user's answer is contained in the actual answer. Another option is to provide multiple answers for each question and check by iterating (`foreach`) through them to see if any match.

- Another way to improve the answer checking is to transform the quiz so that it is multiple choice. You'll need an additional list to hold the answer choices for each question. The possible answers will be a list of lists, with each sublist holding the answer choices for a particular question. Use the `ListPicker` component to give the user the ability to choose an answer. You can read more about lists in Chapter 19.

Summary

Here are some of the ideas we covered in this tutorial:

- Most fairly sophisticated apps have data (often stored in lists) and behavior—its event handlers.
- Use an `ifelse` block to check conditions. For more information on conditionals, see Chapter 18.
- The blocks in event handlers should refer only abstractly to list items and list size so that the app will work even if the data in the list is changed.
- Index variables track the current position of an item within a list. When you increment them, use an `if` block to handle the app's behavior when the user reaches the end of the list.