|  |  |
| --- | --- |
| The Quiz App presents a quiz about pioneers in computer science.The quiz uses lists, labels, images, and a button to create a simple question and answer quiz that displays ‘Correct!’ or ‘Incorrect!’ depending on the user’s answer. Indexing is used to keep track of the current question and access its correct answer and corresponding image.  **Objectives:** In this lesson you will learn to:   * navigate through a list using an index variable; * select items from a list; * check for the end of the list; * work with parallel lists.   **Quiz Content Resources:**   * <https://en.wikipedia.org/wiki/Alan_Turing> * <https://www.nasa.gov/modernfigures> * <https://en.wikipedia.org/wiki/Grace_Hopper> | ***[Click to watch preview video](https://youtu.be/wtpHumLCbVY).*** |

## 

# Getting Ready

Open App Inventor with the [Quiz App Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit6/templates/QuizApp/QuizAppTemplate.asc) . The template app just has the image files for the app, but no design or code yet.

# 

# 

# Quiz App User Interface

## 

The UI for our Quiz app will consist of five types of *Components,* a *Button*, an *Image*, a *TextBox,* some *Labels* and a *Horizontal Arrangement.*

The image is used to show a picture that corresponds with the current question.

The label under the image is the question label which is used to display the current question.

The *TextBox* is where the user enters an answer. The user submits the answer by clicking the answer button next to the Textbox.

The Button to the right is the next button. This is used to display the next question with its corresponding picture. When the user reaches the end of the quiz, the next button restarts the quiz by displaying the first question and picture again.

The bottom label displays a ‘Correct!’ or ‘Incorrect!’ message depending on whether or not the user selects the correct answer to the current question.

### Adding the Image

Drag and drop an image component into the Viewer. You can leave the image without a picture for now or put in one of the images. Later you will be using the rest of the pictures that are included in the template in your program.

### Adding the Labels

Add two labels to the Viewer. Rename the first label ‘QuestionLabel’ and the second label ‘CorrectLabel’. Change QuestionLabel’s text to say ‘Question 1’ and change CorrectLabel’s text to be the empty text. These are just placeholders for now and will be changed later. Feel free to modify the font, font size, etc. of each label.

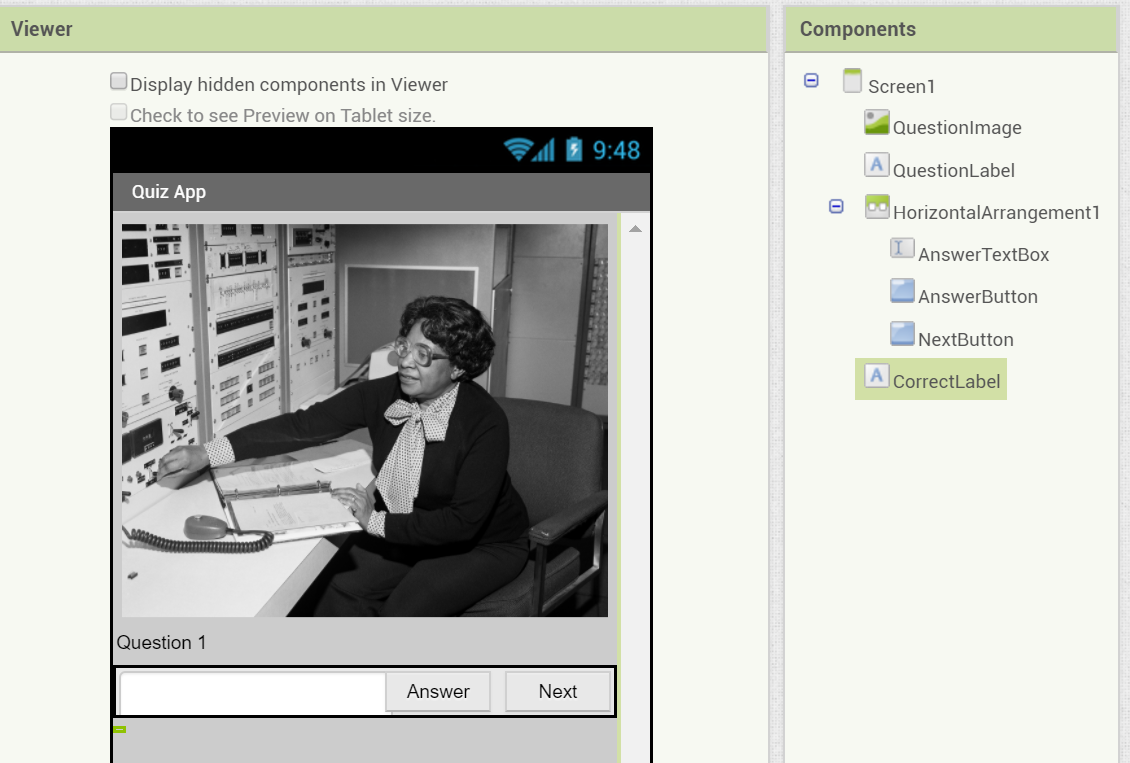
### Add the AnswerTextBox and AnswerButton

Drag and drop a HorizontalArrangement from the Palette’s Layout drawer, then drag and drop a TextBox and Button into it. Rename the TextBox to ‘AnswerTextBox’ and the button to ‘AnswerButton’. Change the hint of the TextBox to “please enter an answer” and change the text of the button to “Submit”

### Adding the Next Button

Add a button to the Viewer. Rename the button ‘NextButton’ and change it’s text to say ‘Next’.

Your finished Screen1 should look something like this:



Note that because the initial value of the CorrectLabel is the empty text, it appears as a small greenish box at the bottom of the screen.

## Coding the Behavior

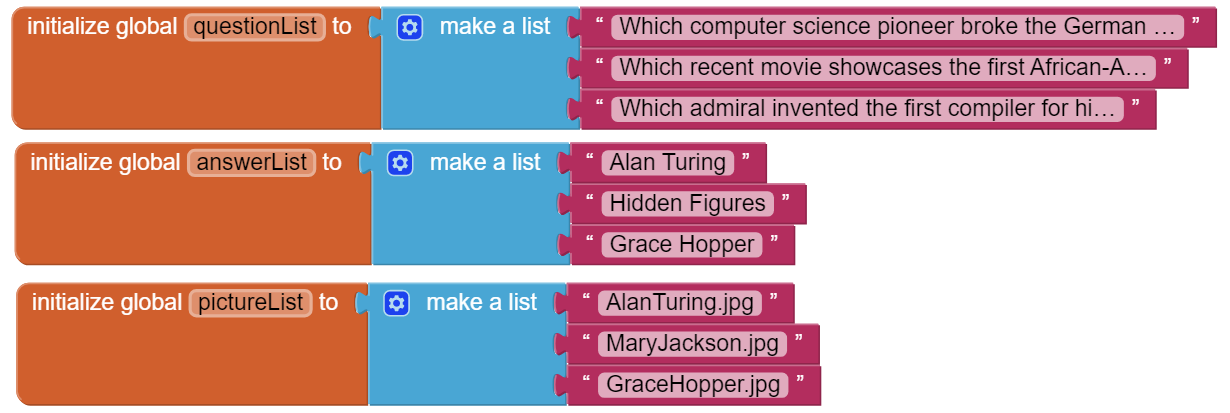
# The Quiz app is an interactive quiz where the user answers a series of questions about various people. With each question the app reports whether the user’s answer is correct or incorrect. The user can move on to the next question by clicking the Next button which causes a new question and its corresponding picture to appear. Questions, their correct answers, and pictures are stored in three separate lists. *Indexing* is used to make sure that a particular question is matched with its correct answer and correct picture.

### Creating the questionList, answerList, and pictureList

One of the most basic types of data that computers have to deal with are *lists.* This app requires three lists:

1. a list of questions
2. a list of answers
3. a list of pictures

The picture files have already been added in the starter app. But you’ll need to create three lists by dragging out initialize global blocks from the Variables drawer, make a list blocks from the lists drawer, and text blocks from the text drawer. Plug the blocks together and enter text for the questions and answers so that the list variables look like the following:



The questions we used are:

1. Which computer science pioneer broke the German Enigma Code during the World War II?
2. Which recent movie showcases the first African-American women who worked as human “computers” for NASA?
3. Which Navy admiral helped create COBOL, one of the first high level programming languages?

The matching answers are:

1. Alan Turing
2. Hidden Figures
3. Grace Hopper

The matching image files in the template are:

1. AlanTuring.jpg
2. MaryJackson.jpg
3. GraceHopper.jpg

Notice that the first question in **questionList** goes with the first answer in **answerList** and those go with the first picture referenced in **pictureList**. The same format is used for the other three questions. These three lists are examples of ***parallel lists*** because the first item on each list are related to the question 1, the second items are related to question 2, and so forth. This parallel setup allows you to easily keep track of questions, their answers, and their pictures using *indexing*. In addition, following this setup allows you to put in your own questions, answers, and pictures without ruining the functionality of the entire app!

**Initializing an Index Variable**

List indexing is an important part of programming. It allows programmers to easily reference items that may have been stored within a list. To keep track of the current question, define and initialize a global variable ***index*** with a value of 1. \Screen Shot 2014-07-28 at 12.38.33 PM.png

## Handling the Screen Initialize Event

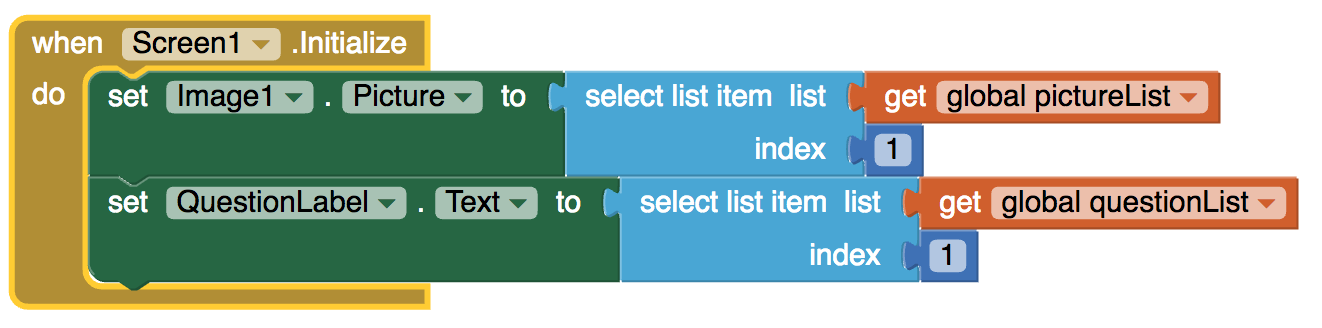
Begin by getting a *Screen1.Initialize* event handler from the Toolbox. The *Screen1.Initialize* event can be used to implement code when the app is started. For this app:

1. Set *QuestionLabel.Text* to the first item in the list *questionList.* The first item in the list is represented by using the *select list item* block from the Lists drawer, then specifying the name of the list in the *list* slot, and plugging a number 1 block into the *index* slot like below. We’ll talk more about indexing in the next section.



2. Set *Image1.Picture* to the first item in the list *pictureList*. This is the same as above, except the name of the list has changed.

The completed handler should look like this:



## Running and Testing the App

## This would be a good time to test the app, making sure it correctly displays the first quiz question and corresponding picture.

## Handling the NextButton Click Event

The code you have so far works only for the first question of the quiz. Now, let’s code the app so that the other three questions will appear, one each time the Next button is clicked. To begin, get a *NextButton.Click* event handler from the Toolbox.

Now, when the user clicks the Next button the following things should happen:

1. ***index***should be *incremented* to move to the next question. Incrementing a variable means that you should add 1 to it using a get block and save it back in the variable using a set block.

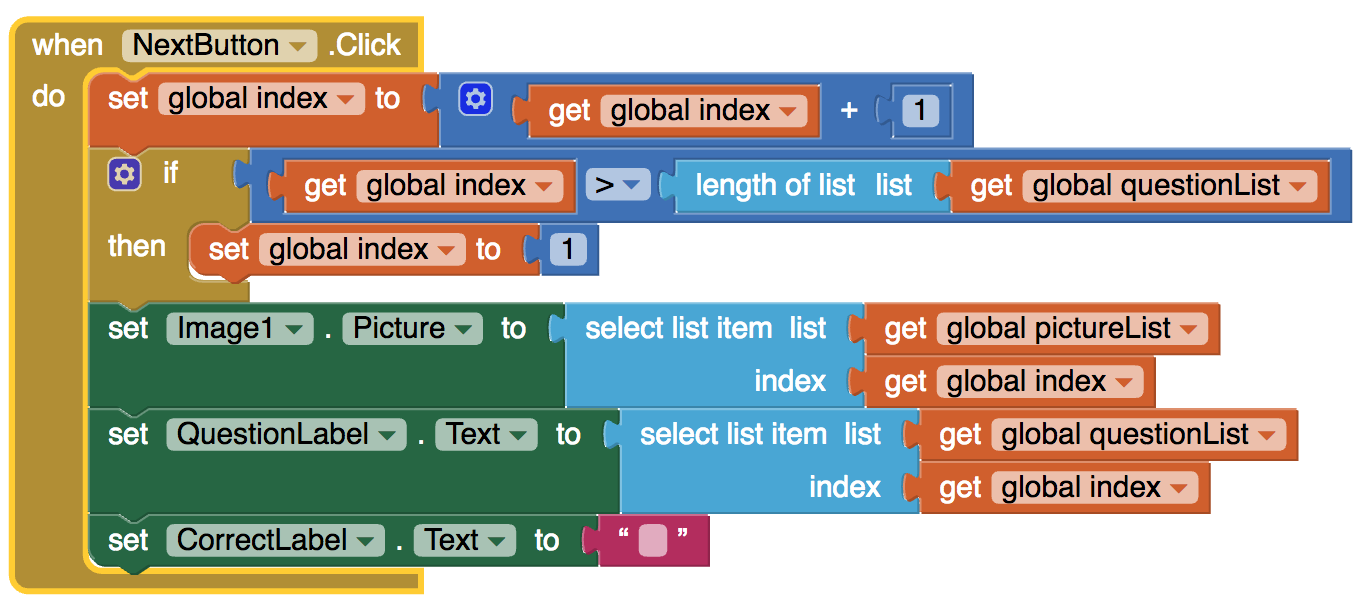


1. What happens when the user reaches the last question (in this case question number 3)? We want to move the index back to question 1. You will need an if statement to check for this. Note that you should compare to a ***length of list***block instead of “3” so that your quiz will still work even if you add more questions.
2. *QuestionLabel*’s text should be set to the question at the *index* in *questionList* using a *select list item* block with the *index* variable. When index is 1, this will select the first question. When index is 2, it will select the 2nd question, and so on.



1. *Image1*’s picture should be set to the picture at the *index* in *pictureList* using a *select list item* block similar to the code above.
2. *CorrectLabel*’s text should be set to the empty text block so it no longer shows whether the previous answer was correct or not.

When completed, the next button handler should look like this:



## Running and Testing the App

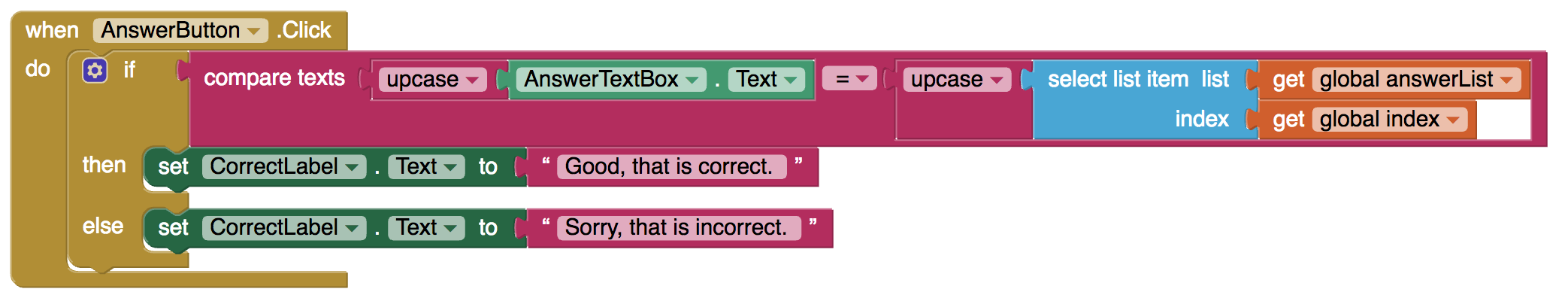
Run the app again and test that you can cycle through all 3 questions and that the algorithm correctly wraps around to the beginning of the list.

## Checking the User’s Answer

The user answers a question by entering it in AnswerTextBox and clicking the AnswerButton.

When the AnswerButton is clicked, check if the answer is correct using an If block. You can use the blue mutator button in the if block to add an else. In the if condition, compare the user’s typed in answer (AnswerTextbox.Text) with the correct answer in the *answerList* by using a ***select list item*** block and the index. If the user chooses the right answer, display ‘Correct!’ in the correctLabel. Otherwise, display ‘Incorrect!.

When complete, this is how your code should look -- note the use of the ***compare texts*** block from the *Text* drawer. You could also use the equals operator from the Math drawer.



## Running and Testing the App

That’s it. When you run and test the app, it should work correctly for this 3-questions quiz.

## Enhancements and Extensions

Here are some programming problems that will let you enhance and extend the Quiz App.

1. As you might have noticed, if the answer is “Alan Turing” and the user types in “alan turing”, the answer will be marked incorrect. That’s not very nice for the user. To remedy this problem you will want to convert both the user’s answer and the stored answer to upper case “ALAN TURING”. (HINT: use the ***upcase*** block in the *Text* drawer to convert both strings.)
   1. When the user gets an incorrect answer, instead of just reporting “incorrect”, use a join block to also display the correct answer. “Sorry, that is incorrect. The correct answer is Grace Hopper.”
2. **Random Question.** Add *RandomButton* to the app that when clicked will display a random question from the quiz. (HINT: You could use some new blocks from the List drawer such as a pick a random item block fed into a index in list thing block to set the index randomly.)
3. Add a fourth question (and answer and picture) to the quiz. If you like, you can research “famous computer scientists” on the Web to discover a fourth person. Or, if you wish, you can create a question about Hal Abelson, the creator of our App Inventor programming language, pictured on the right. (HINT: You should only have to modify the 3 lists and upload an image file. The code should work with any number of questions as long as you used the ***length of list*** block instead of hard coding in the number 3 for the number of questions.)

## Reflection for the student

In your portfolio, create a new page named ***Quiz App*** and answer the following questions:

1. Describe the significance of the global variable index. How is indexing used with lists in this app?
2. Describe how parallel lists were used in this app. Why was the parallel structure of the lists necessary?
3. Include screenshots of your code for exercises 2 and 3 from the *Enhancements* section.
4. Include a screenshot of the code that added your extra question (exercise 4). Explain why the code for the buttons worked without any changes after the addition of the extra question.

***Nice work! Complete the Self-Check Exercises and Portfolio Reflection Questions as directed by your instructor.***