

CHAPTER 7

Lists

starting out with >>> **APP INVENTOR**
FOR ANDROID



TONY GADDIS · REBECCA HALSEY

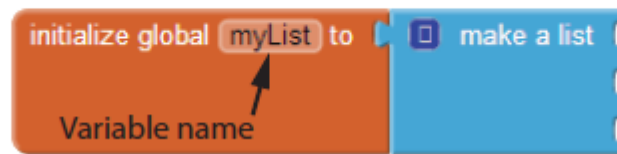
Topics

- Creating a list
- Iterating over a list with the `for each` loop
- Selecting an item
- Inserting and appending items
- Removing items
- Replacing items
- Searching for an item
- Other list operations

Creating a List

- A list is a single object that contains multiple items of related data.
- To create a list, first need to create a *variable*.
- The variable will hold the list of multiple items.
- To create a list by plugging the `make a list` block into the list variable.
- The `make a list` method is located in the *List* drawer.

Figure 7-1 Create a Variable that Holds a List (Source: MIT App Inventor 2)



Creating a List

- Next, you can begin adding items to your list.
- To add a text item to your list, drag a text block (from the *Text* drawer).
- Change the value to the data that you wish to add and plug it in.

Figure 7-2 Add a Text Item to a List (Source: MIT App Inventor 2)



Creating a List

There are two steps to make your list visible.

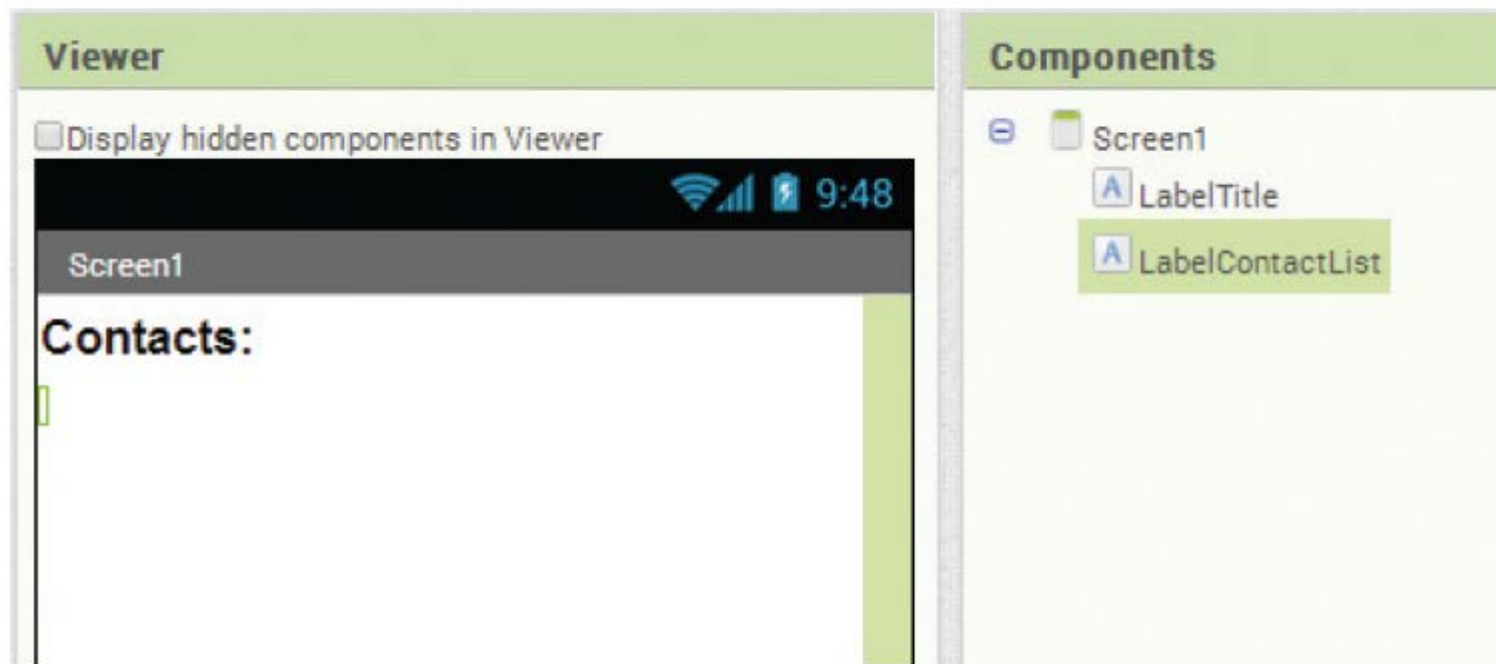
1. Use a component such as a Label to display your list.
2. You must have an event that populates the Label once the list is created.

The `Screen.Initialize` event will work if you want to show the list when your application loads.

Creating a List

Figure 7-3 shows the app in the Designer.

Figure 7-3 Create a List–Design View (Source: MIT App Inventor 2)



Creating a List

- In Figure a 7-4, we have created a list of names using the `make a list` block.
- We put the names in simple text blocks, and plug those into the `make a list` block.
- Store the entire list in a variable named `ContactList`.
- We use the `Screen1.Initialize` event to set the `LabelContactList.Text` property to the value of the `ContactList` variable.
- Using the `Screen.Initialize` event, we populate the three names as soon as the application loads.

Creating a List

Figure 7-4 Creating a List–Blocks Editor (Source: MIT App Inventor 2)



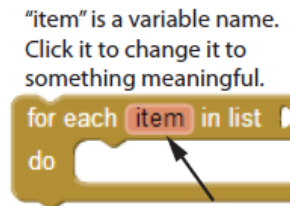
Iterating Over a List with the `for each` Loop

Iteration means to repeat the same process over and over until you reach the result you're looking for. To iterate a list generally means to step through all of the list items, one at a time, until you reach the end.

Iterating Over a List with the `for each` Loop

- The `for each` loop is designed to work with a list.
- When the loop executes, it iterates once for each item in the list.

Figure 7-16 The `for each` LOOP (Source: MIT App Inventor 2)



- The `for each` block has a variable named `item` after the words “for each”.

Iterating Over a List with the `for each` Loop

The `for each` loop executes in the following manner:

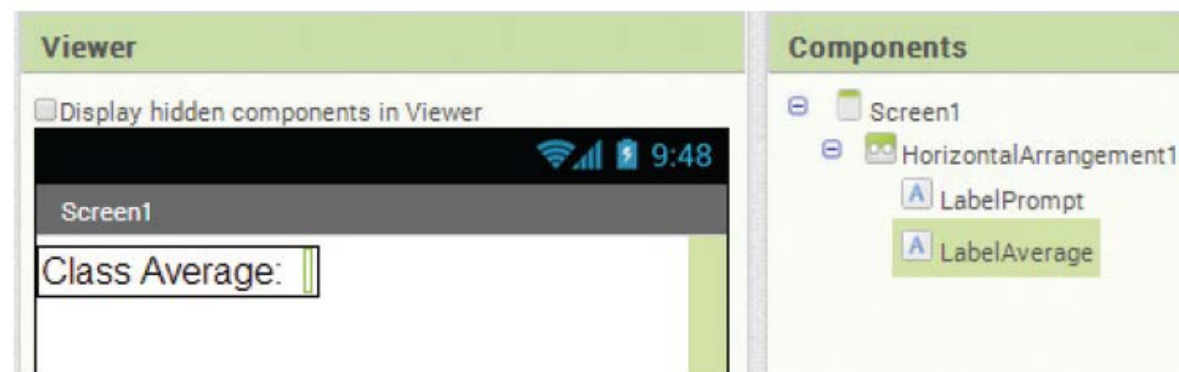
- The `item` variable is assigned the first value in the list.
- The blocks that appear inside the `for each` block are executed.
- The `item` variable is assigned to the next value in the list.
- The blocks that appear inside `for each` block are executed again.
- This continues until the `item` variable has been assigned the last value in the list.

Iterating Over a List with the `for each` Loop

Test Scores Example

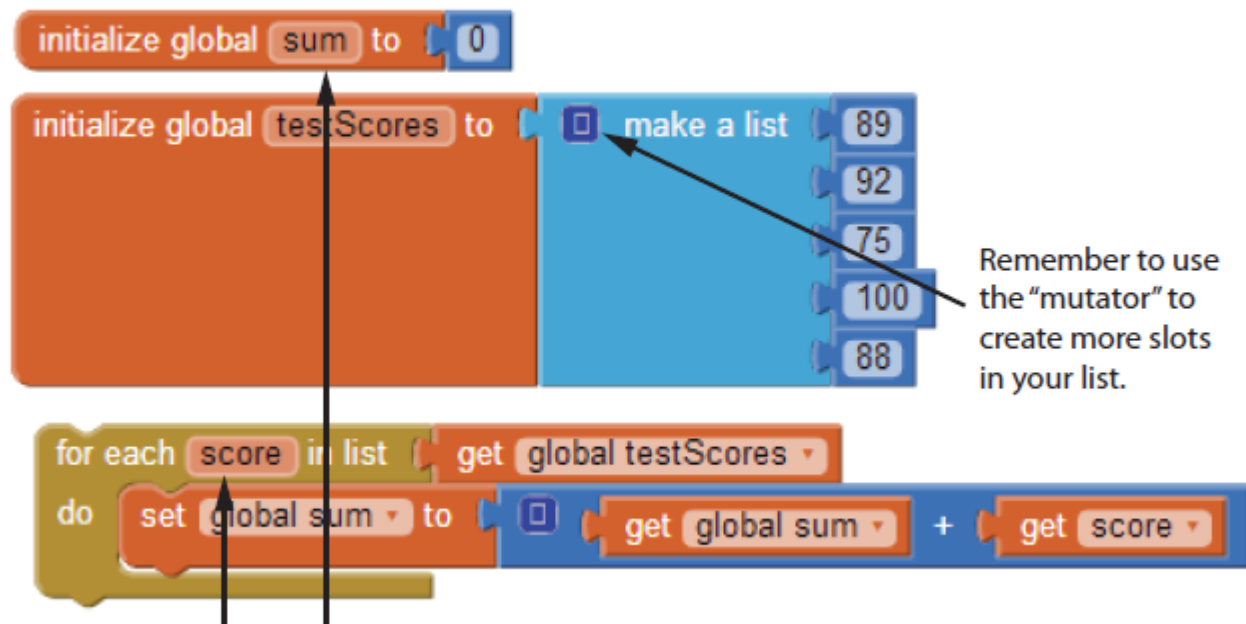
- In the Test Scores example, a list of a class's test scores is created and stored to a variable named `TestScores`.
- We will use a `for each` loop to iterate through the list and accumulate the sum of the scores and calculate the class average.

Figure 7-17 Test Scores App Design (Source: MIT App Inventor 2)



Iterating Over a List with the for each Loop

Figure 7-18 Test Scores Code in the Blocks Editor (Source: MIT App Inventor 2)



Remember to use the "mutator" to create more slots in your list.

Remember to hover over a variable to get their "get" and "set" blocks.

Iterating Over a List with the `for each` Loop

Test Scores Example

1. We create a variable `testScores` to hold the list.
2. We make the list and populate the scores with `number` blocks rather than `text` blocks.
3. We create a variable `sum` to hold the sum of numbers.
4. The `for each` block has an `item` variable and also requires a block, which is the variable that holds a list, to be plugged into it.
5. The block `get global testScores` represents the list that holds the scores, and its “get” block is plugged into the `for each` block.

Iterating Over a List with the `for each` Loop

- After the `for each` iterates through the list, the `Sum` variable will equal all the test scores added together.
- Last, we will need to divide the sum by the number of tests and place that back into the `Text` property of the label as in Figure 7-19

Figure 7-19 Calculating the Average (Source: MIT App Inventor 2)

```
initialize global sum to 0
initialize global count to 0
initialize global testScores to [make a list [89, 92, 75, 100, 88]]

when Screen1.Initialize
do
  for each score in list [get global testScores]
  do
    set global sum to [get global sum] + [get score]
    set global count to [get global count] + [1]
  do
  set LabelAverage.Text to [get global sum] / [get global count]
```

The image shows a sequence of code blocks in MIT App Inventor 2. At the top, there are two 'initialize global' blocks: 'sum' set to 0 and 'count' set to 0. Below these is another 'initialize global' block for 'testScores', which is set to a 'make a list' block containing the numbers 89, 92, 75, 100, and 88. The main logic is contained within a 'when Screen1.Initialize' block. Inside this block, there is a 'do' block containing a 'for each' loop. The 'for each' loop iterates over the 'testScores' list. Inside the loop, there are two 'do' blocks: the first sets 'global sum' to the current value of 'global sum' plus the current 'score', and the second sets 'global count' to the current value of 'global count' plus 1. After the 'for each' loop, there is a final 'do' block that sets the 'Text' property of a label named 'LabelAverage' to the value of 'global sum' divided by 'global count'.

Iterating Over a List with the `for each` Loop

To complete this example:

1. We added a variable named `count`, for the count.
2. We plugged the `for each` loop into the `Screen1.Initialize` event.
3. We added a statement to count by one in each iteration and assigned the results to the `count` variable.
4. We set the results of the some variable divided by the `count` variable to the `LabelAverage.Text` property.

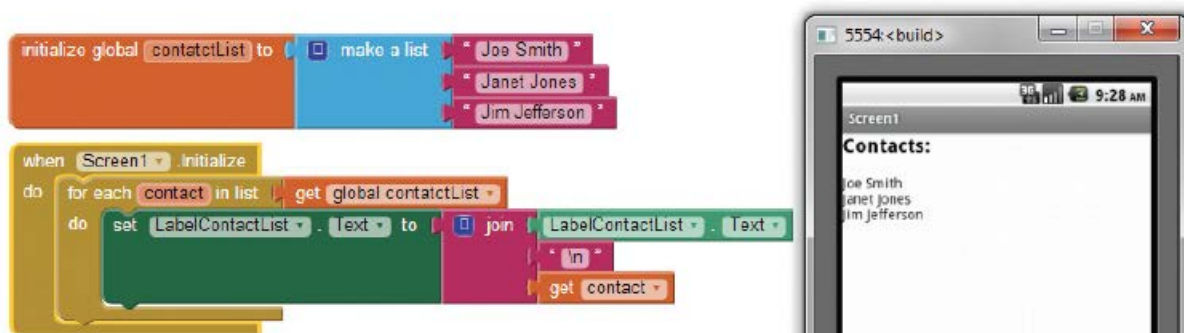
Iterating Over a List with the `for each` Loop

Contact List Example

We are going to print out each list item's value with the return character `\n`, one at a time.

The backslash (`\`) is also known as an *escape sequence*.

Figure 7-20 Adding the Carriage Return (Source: MIT App Inventor 2)



Use the `join` text block to add the item plus the return character `\n` to the label's `Text` property.