1. Variables

1.1 Defining Variables

You define a variable when your app needs to remember something that is not being stored within a component property. For example, a game app might need to remember what level the user has attained.

Variables are defined explicitly in the Blocks Editor by dragging out an initialize global block. You can name the variable by clicking the text "name" within the block, and you can specify an initial value for it by dragging out a number, text, color, or make a list block and plugging it in.

When you define a variable, you instruct the app to set up a named memory slot for storing a value. The number block you plug in specifies the value that should be placed in the slot when the app begins. Besides initializing with numbers or text, you can also initialize the variable with a make a list or create empty list block. This informs the app that the variable will store a list of memory slots instead of a single value.

Here are the steps you'd follow to create a variable called score with an initial value of 0:

1. In Built-in blocks, open the Variables drawer and drag out the initialize global block.

2. Change the name of the variable by clicking the text "name" and typing "score".

3. From the Math drawer, drag out a number block and plug it into the open socket of the variable definition to set the initial value.

1.2 Setting and Getting a Variable

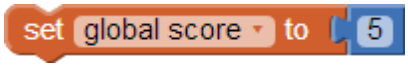
When you define a variable, App Inventor creates two blocks for it: set and get. You can access these blocks by hovering over the variable name in the initialization block. The initialization block contains set and gets blocks for that variable

The set global to block lets you modify the value stored in the variable. The term "**Global**" in the set

global score to block refers to the fact that the variable can be used in all of the program's event handlers and procedures. With the newest version of App Inventor, you can also define variables that are "**Local**" to a particular procedure or event handler. Local variables can be used only by the procedure or event with which they're associated.

1.2.1 Setting a Variable to an Expression

You can put simple values such as 5 into a variable, but often you'll set the variable to a more complex expression. You'll build such expressions with a set of blocks that plug into a set global to block.



1.2.2 Incrementing a Variable

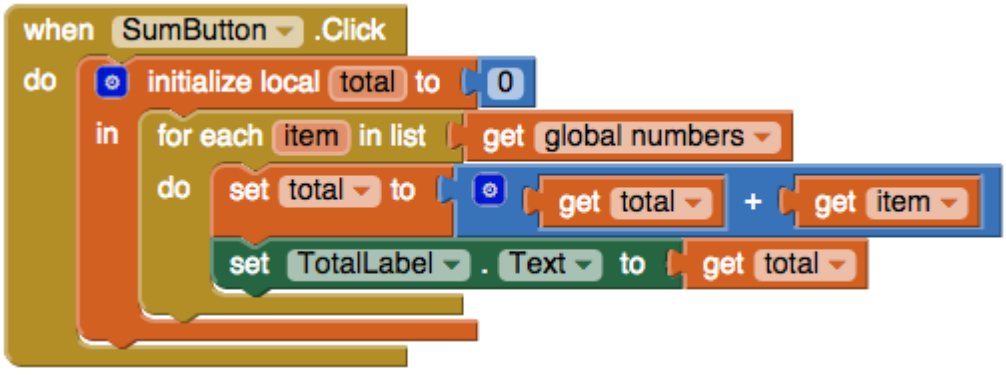
Perhaps the most common expression is for incrementing a variable, or setting a variable based on its own current value. For instance, in a game, when a player scores a point, the variable score can be incremented by 5.

1.2.3 Building Complex Expressions

In the Math drawer, App Inventor provides a wide range of mathematical functions similar to those you'd find in a spreadsheet or calculator. There are arithmetic operators, blocks for generating random values, and operators such as sqrt, cosine, and sine. You can use these blocks to build a complex expression and then plug them in as the right hand side expression of a set global to block. For example, to move an image sprite to a random column within the bounds of a canvas, you'll configure an expression consisting of a multiply block, a subtract block, a Canvas1.Width property, an ImageSprite1.Width property, and a random fraction block

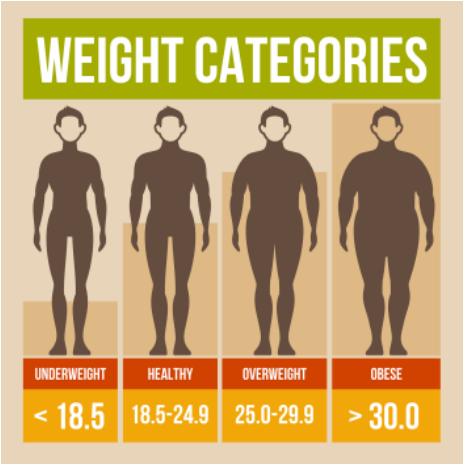
1.3 Local Variables

With the latest version of App Inventor, you can now also define local variables, that is, variables whose scope is restricted to a single event handler or procedure.



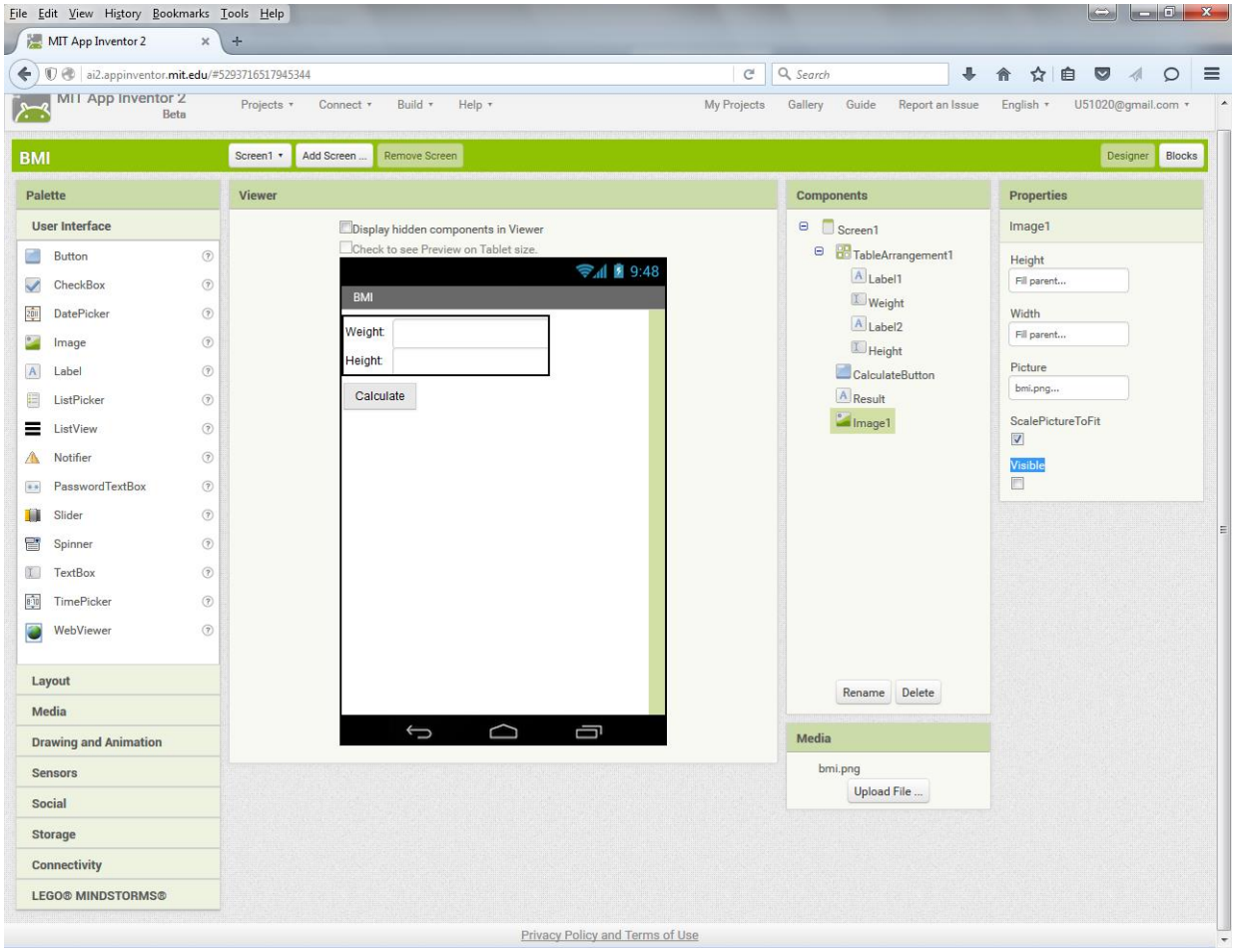
1.4 Exercise: BMI Calculation

1.4.1 Media File



bmi.png

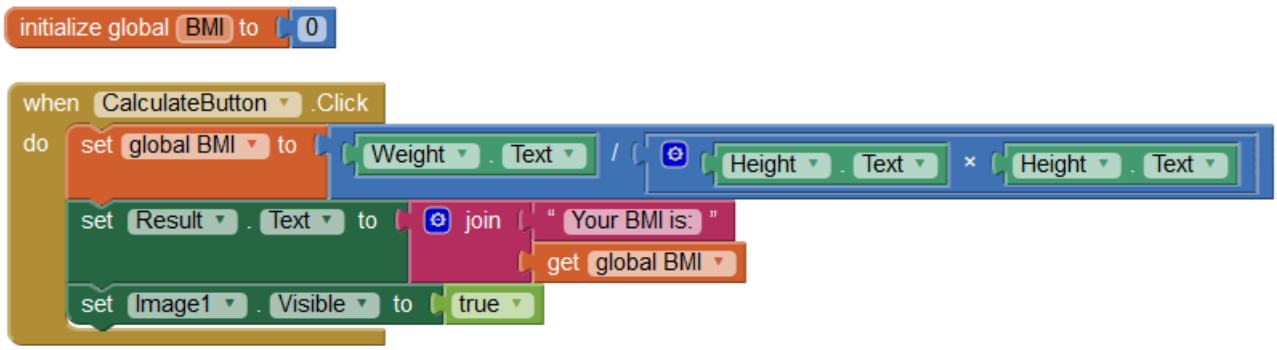
1.4.2 Designer View



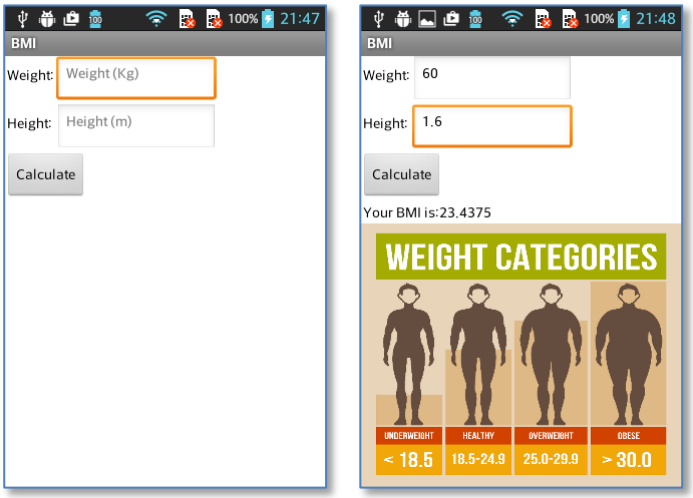
1.4.3 Components

Component	Name	Properties	Remark
Screen	Screen1	Title = "**BMI**"	
TableArrangement	TableArrangement1	Columns = "**2**" Rows = "**2**"	
Label	Label1	Text = "**Weight:**"	Within TableArrangement1
TextBox	Weight	Hint = "**Weight (Kg)**"	Within TableArrangement1
Label	Labe2	Text = "**Height:**"	Within TableArrangement1
TextBox	Height	Hint = "**Height (m)**"	Within TableArrangement1
Button	CalculateButton	Text = "**Calculate**"	
Label	Result	Text = *[Blank]*	
Image	Image1	Picture = "**bmi.png**" ScalePictureToFit = "X" Visible = *[Blank]*	

1.4.4 Block Configuration



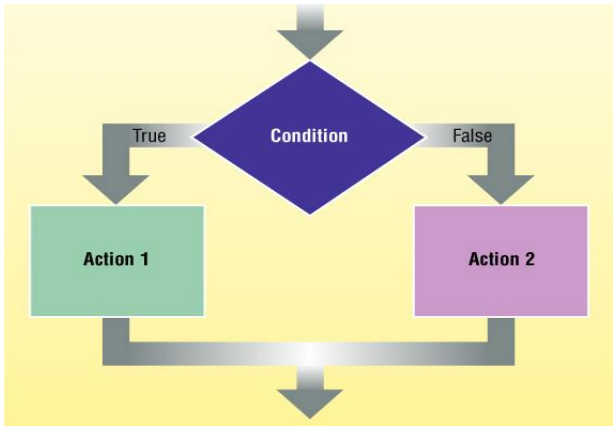
1.4.5 Sample Output

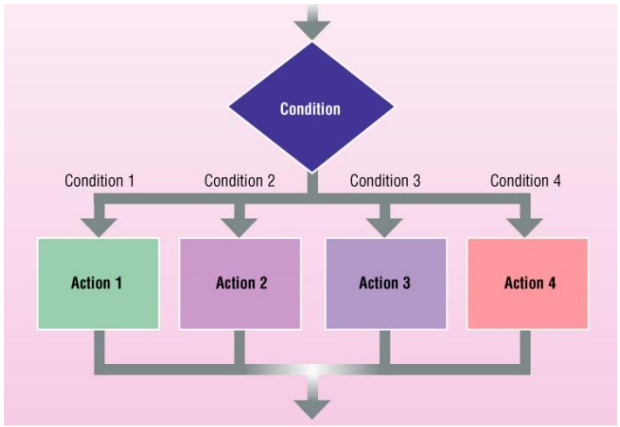


2. Decision Making

2.1 Conditional Blocks

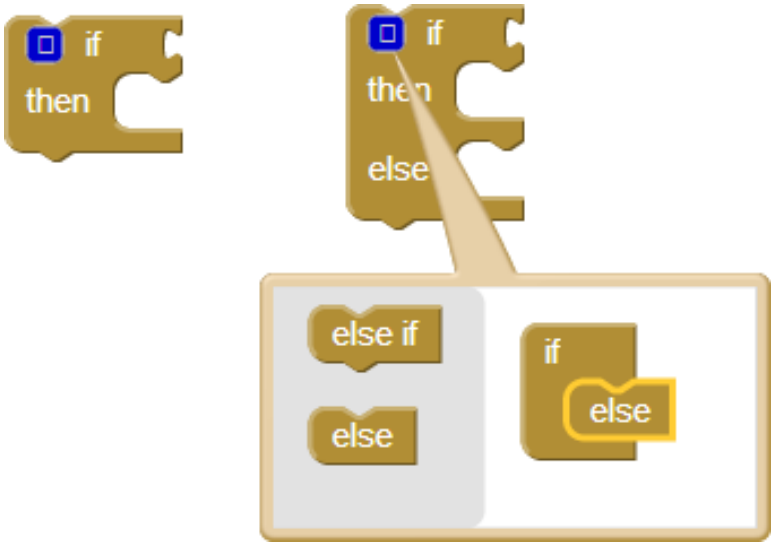
An event handler that tests for a condition and branches accordingly.





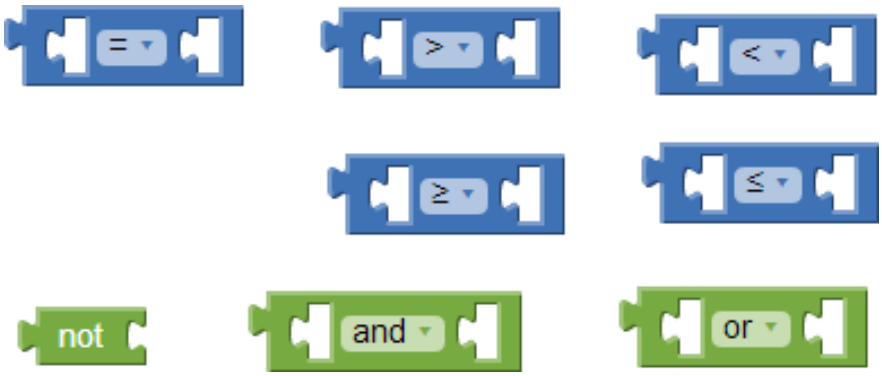
2.2 The if and else if conditional blocks

To allow conditional branching, App Inventor provides an if-then conditional block in the Control drawer. You can extend the block with as many else and else if branches as you'd like by clicking the blue icon



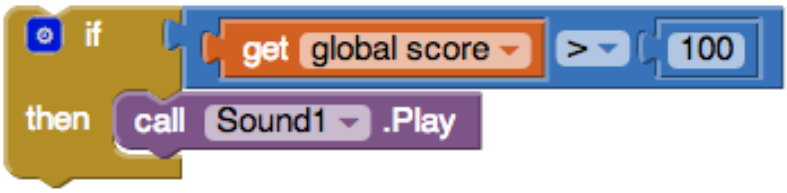
2.2.1 Relational and Logical Operator

You can plug any Boolean expression into the test sockets of the "if and else if" blocks. A Boolean expression is a mathematical equation that returns a result of either true or false. The expression tests the value of properties and variables by using relational and logical operators



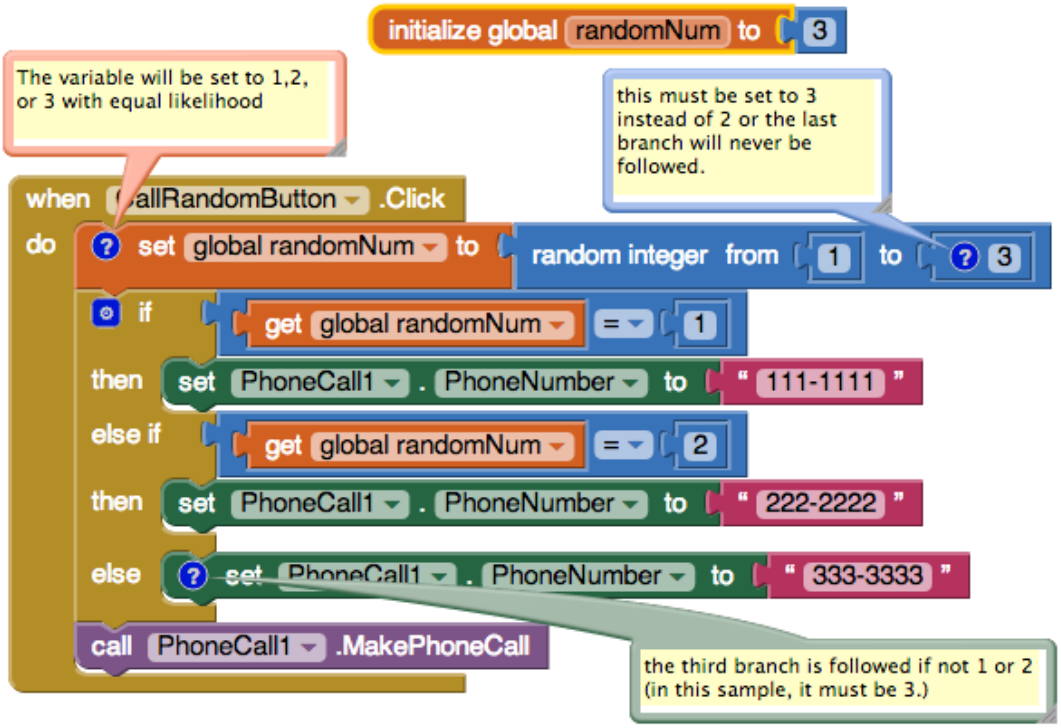
2.2.2 A Boolean Expression

The blocks you put within the "then" socket of an if block will only be executed if the test is true. If the test is false, the app moves on to the ensuing blocks



2.2.3 Programming Conditions within Conditions

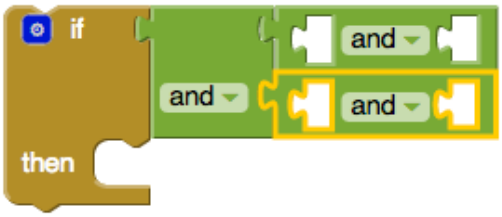
Many decision situations have more than just two outcomes from which to choose. You can add as many else if branches as you like. You can also nest conditionals within conditionals. When conditional tests are placed within branches of another conditional test, we say they are nested. You can nest conditionals and other control constructs such as for each loops to arbitrary levels in order to add complexity to your app.



2.3 Programming Complex Conditions

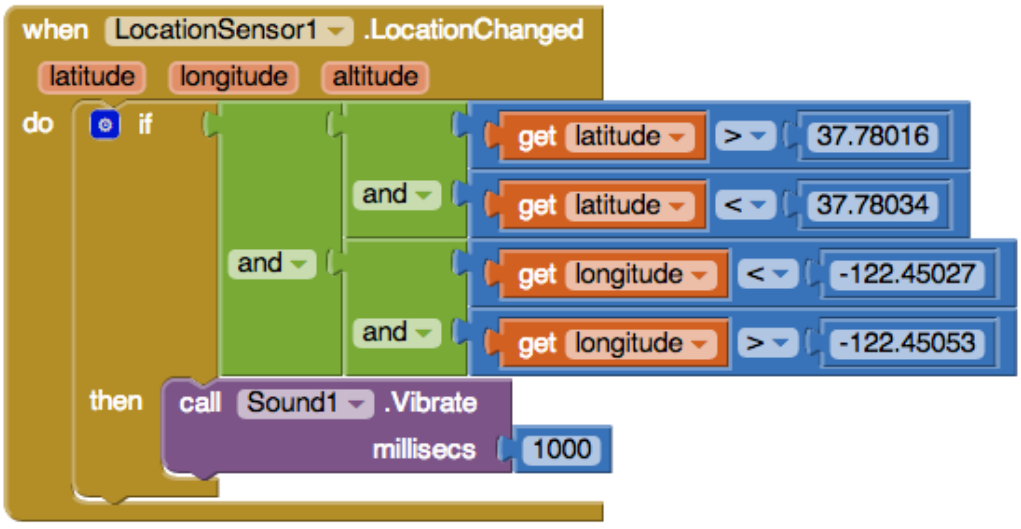
You can build complex tests by using the logical operators and, or, and not, which you can find in the Logic drawer. In this case, you drag out an if block and some and blocks, place one of the and blocks within the “test” socket of the if, and the others within the first and block

2.3.1 An if test can test many conditions using and, or, and other relational blocks



2.3.2 Blocks for the first test are placed into the and block

2.3.3 This event handler checks the boundary each time the location changes

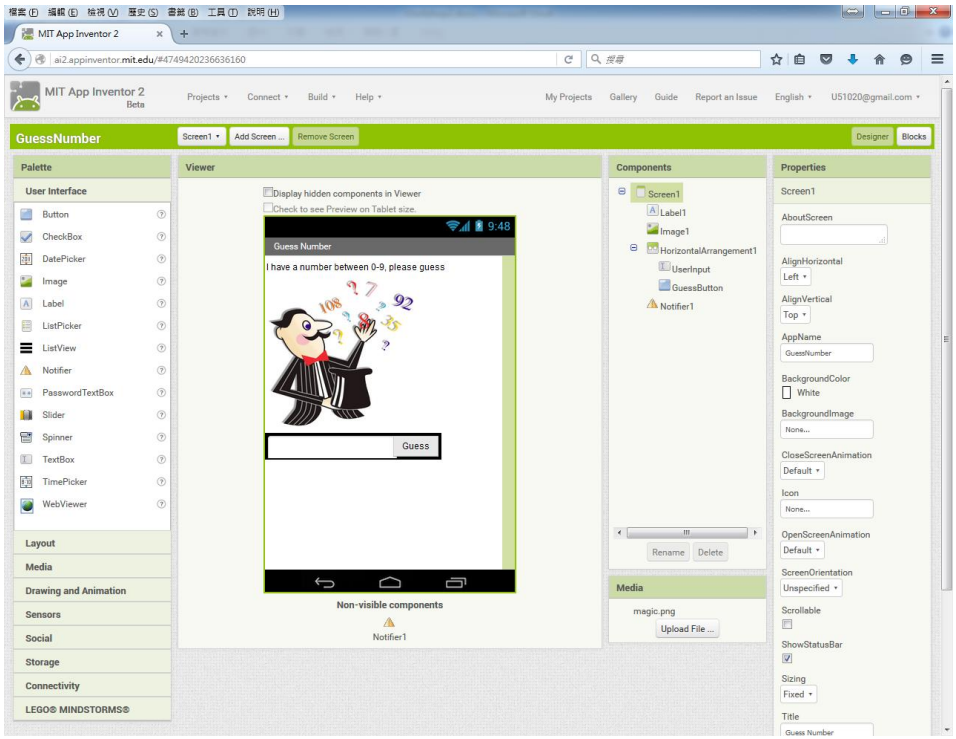


2.4 Exercise: Guess Number

2.4.1 Media File

magic.png

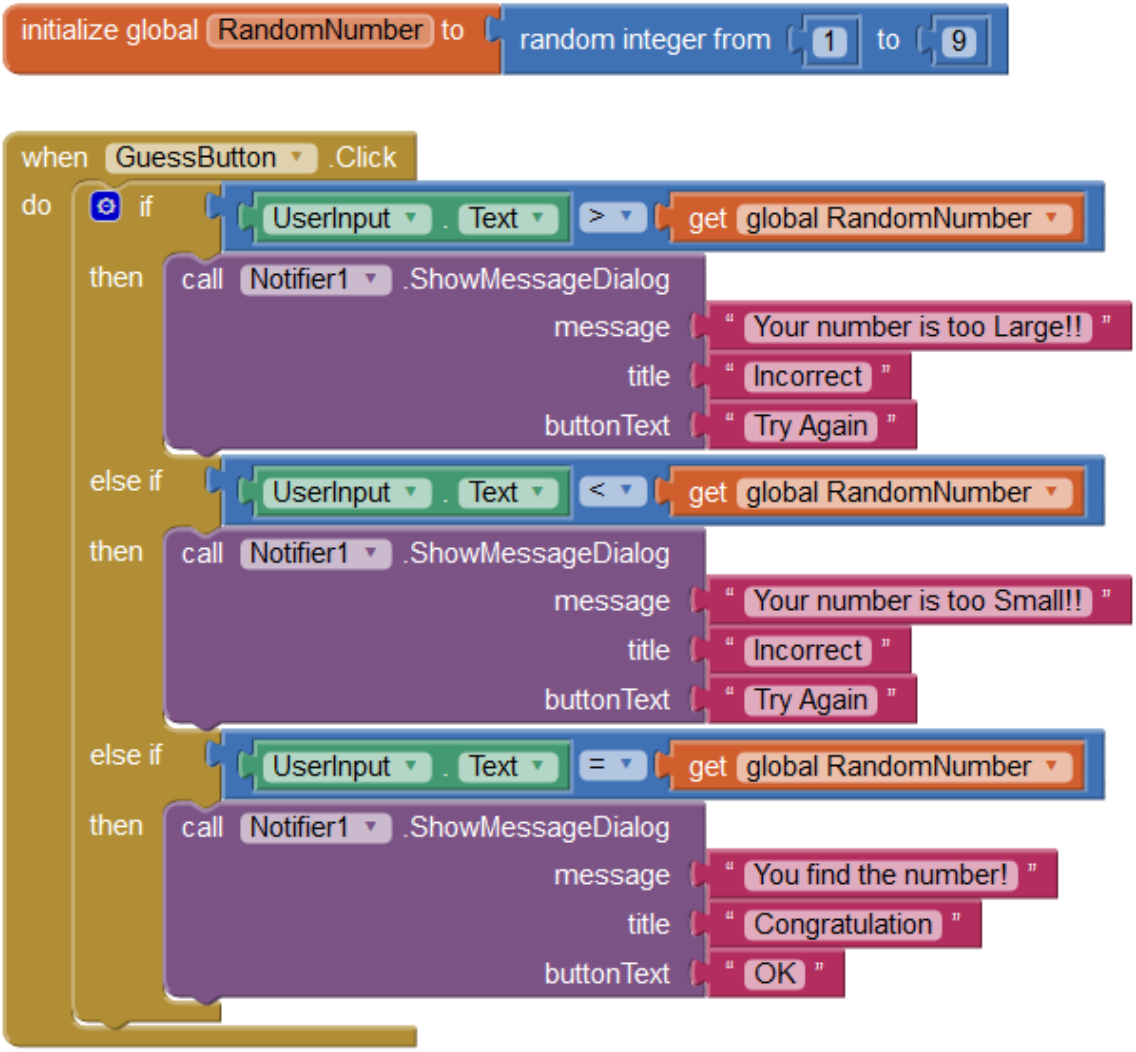
2.4.2 Designer View



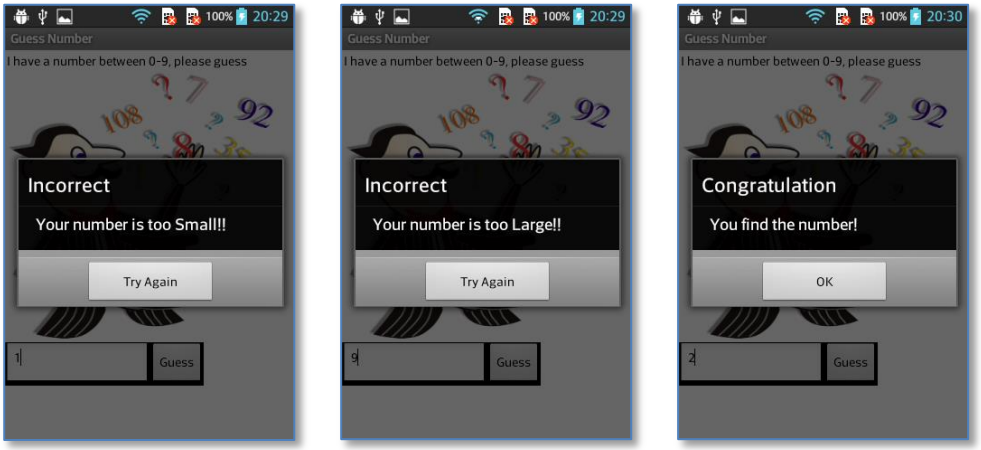
2.4.3 Components

Component	Name	Properties	Remark
Screen	Screen1	Title = "**Random Number**"	
Label	Label1	Text = "**I have a number between 0-9, please guess**"	
Image	Image1	Height = "**300 pixels**" Width = "**300 pixels**" Picture = "**magic.png**"	
HorizontalArrangement	HorizontalArrangement1		
TextBox	UserInput	Hint = "**Your Answer**"	Inside HorizontalArrangement1
Button	GuessButton	Text = "**Guess**"	Inside HorizontalArrangement1
Notifier	Notifier1		

2.4.4 Block Configuration



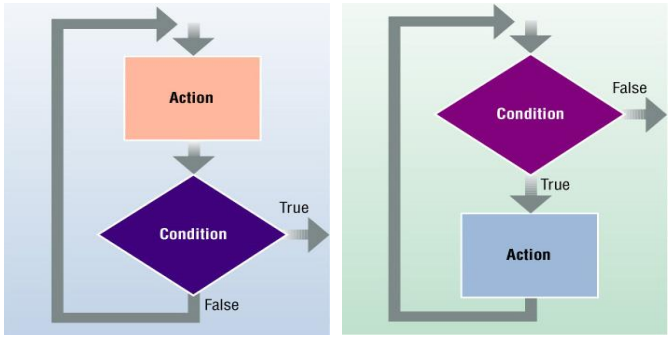
2.4.5 Sample Output



3. Looping

3.1 Overview

Repeat blocks are the other way in which an app behaves in a nonlinear fashion. Just as if and else if blocks allow a program to branch, repeat blocks allow a program to loop; that is, to perform a set of functions and then jump back up in the code and do it again. When an app executes, a program counter working beneath the hood of the app keeps track of the next operation to be performed.
With repeat blocks, the program counter loops back up in the blocks, continuously performing the same operations.
App Inventor provides a number of repeat blocks, including the "**for each**" and "while". "**foreach**" is used to specify functions that should be performed on each item of a list. The while block is more general than the for each. With it, you can program blocks that continually repeat until some arbitrary condition changes.



3.2 The while Loop

The while loop is one of the most common constructs in programming. A while loop is a control structure that allows you to repeat a task a certain number of times

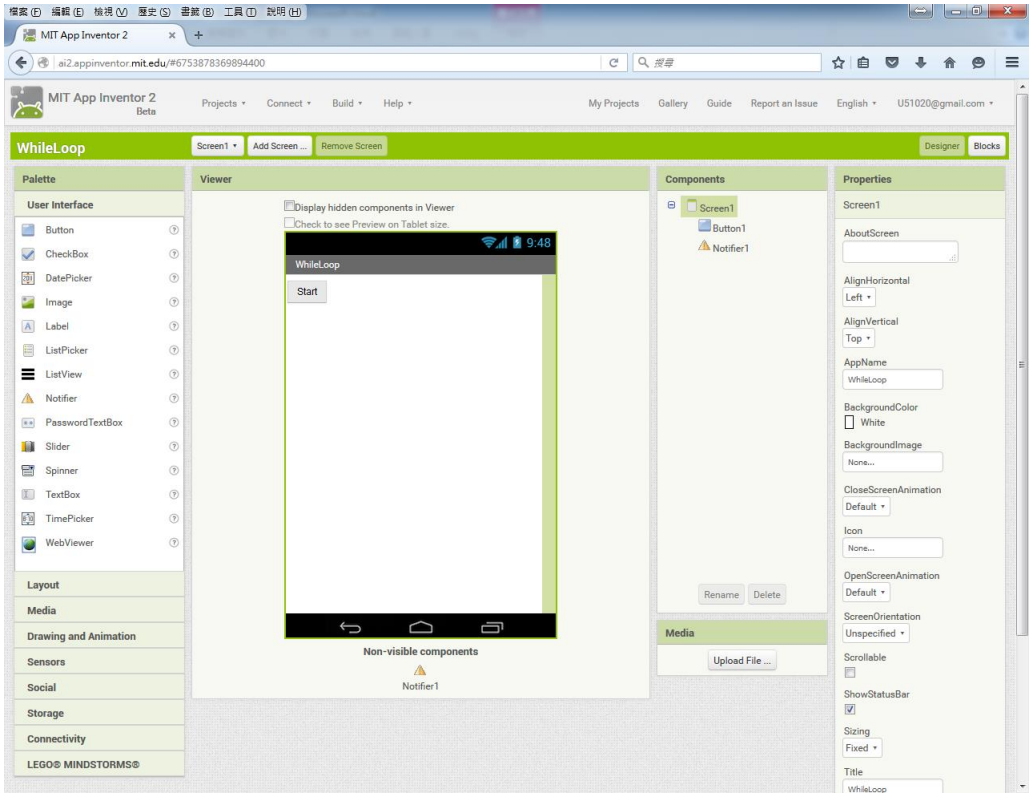
3.3 The for Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. Here is the flow of control in for loop:

1. The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
2. Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables.
4. The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself. After the Boolean expression is false, the loop terminates.

3.4 Exercise: While Loop

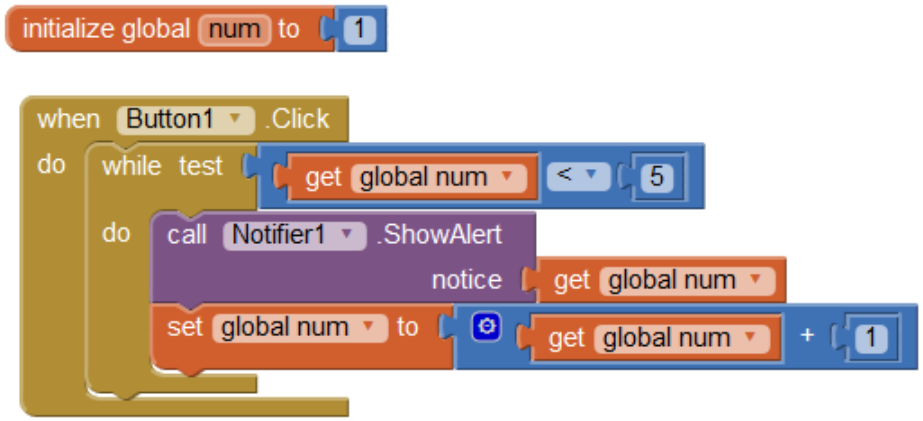
3.4.1 Designer View



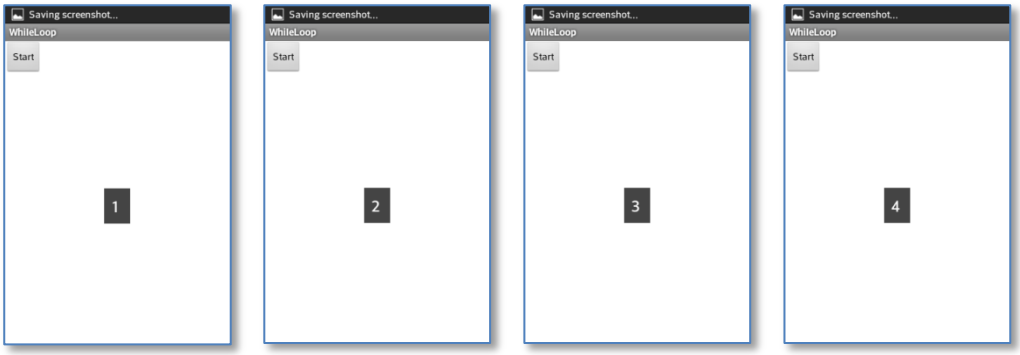
3.4.2 Components

Component	Name	Properties	Remark
Screen	Screen1	Title = "**While Loop**"	
Button	Button1	Text = "**Start**"	
Notifier	Notifier1		

3.4.3 Block Configuration

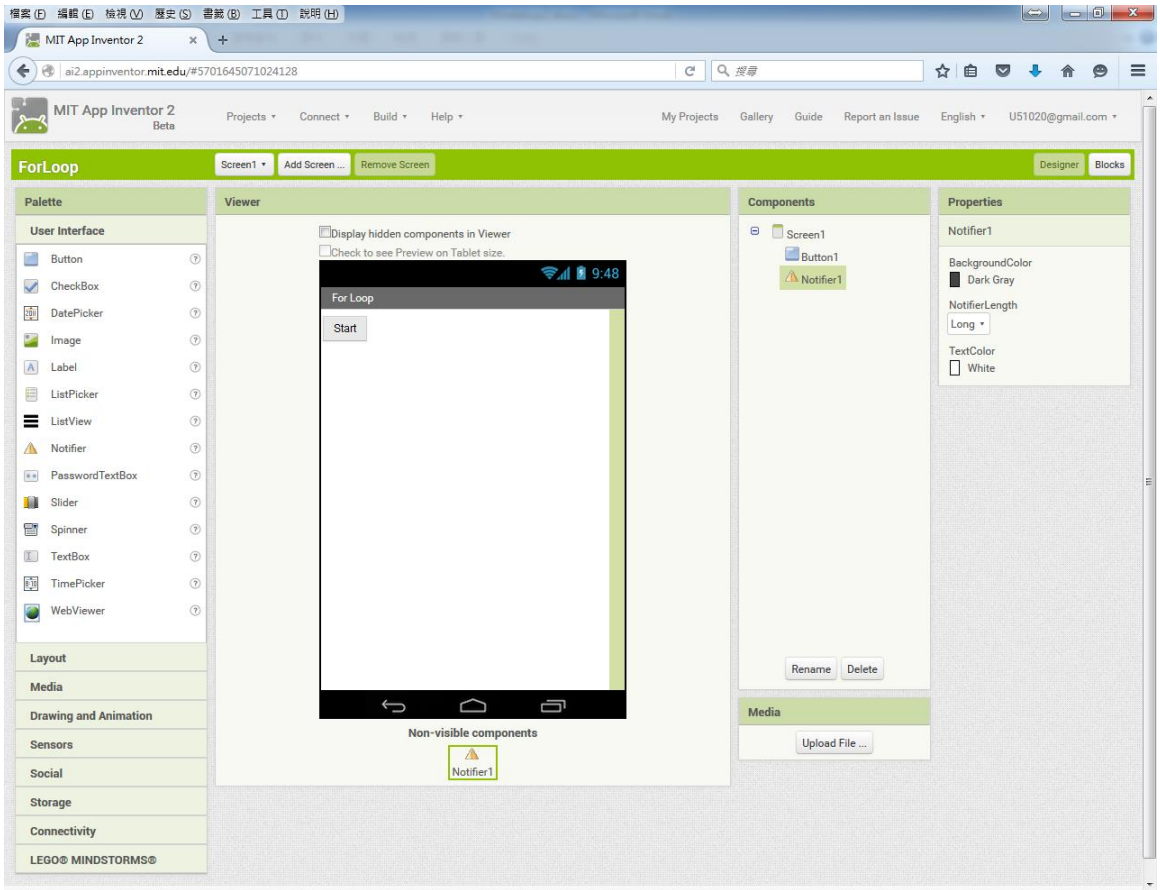


3.4.4 Sample Output



3.5 Exercise: For Loop

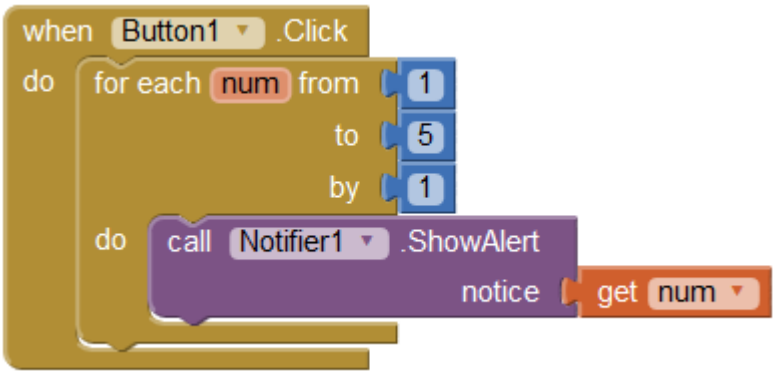
3.5.1 Designer View



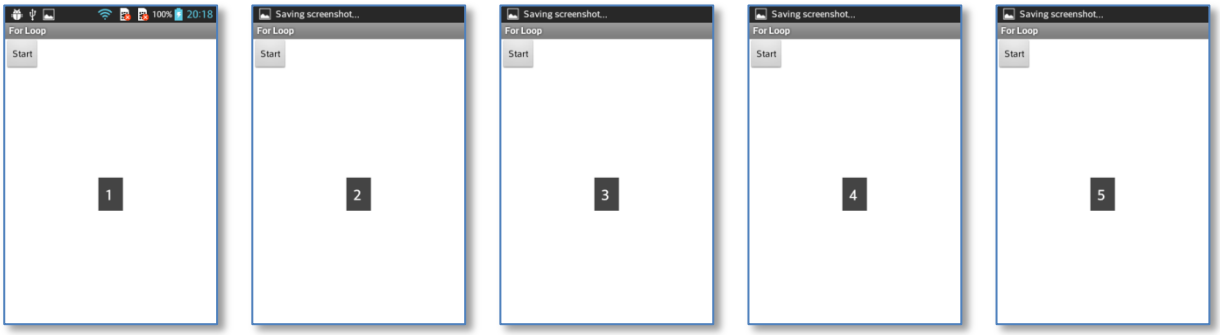
3.5.2 Components

Component	Name	Properties	Remark
Screen	Screen1	Title = "**For Loop**"	
Button	Button1	Text = "**Start**"	
Notifier	Notifier1		

3.5.3 Block Configuration



3.5.4 Sample Output



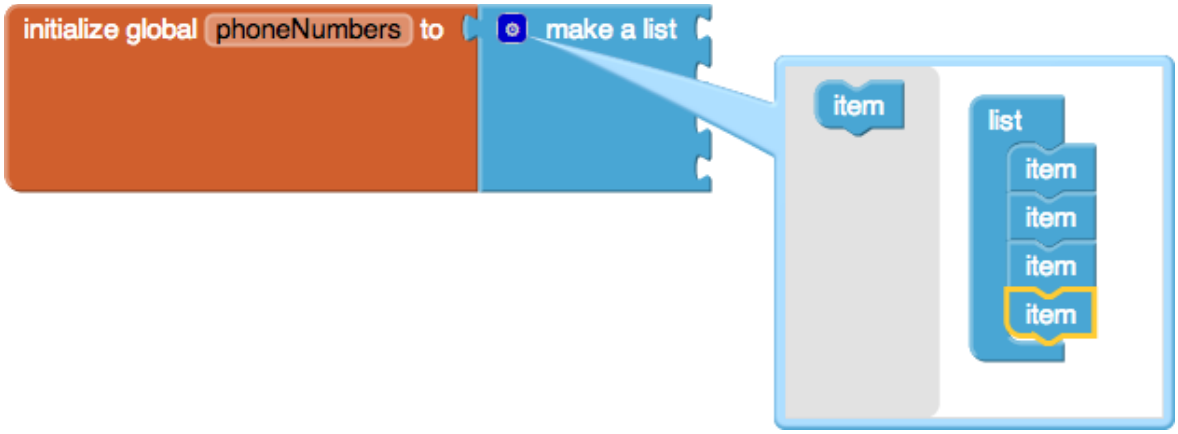
4. List

4.1 Overview

Many apps process lists of data. For example, Facebook processes your list of. You specify list data in App Inventor with a variable, but instead of naming a single memory cell with the variable, you name a set of related memory cells. You specify that a variable is multi-item by using either the make a list or create empty list blocks.

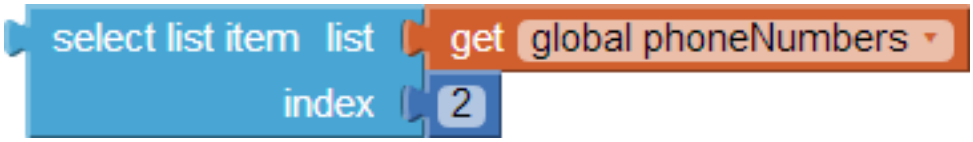
4.2 Creating a List Variables

You create a list variable in the Blocks Editor by using an initialize global variable block and then plugging in a make a list block. You can find the make a list block in the Lists drawer, and it has only two sockets. But you can specify the number of sockets you want in the list by clicking on the blue icon and adding items.



4.3 Selecting an Item in a List

As your app runs, you'll need to select items from the list; for example, a particular question as the user traverses a quiz or a particular phone number chosen from a list. You access items within a list by using an index; that is, by specifying a position in the list. If a list has three items, you can access the items by using indices 1, 2, and 3. You can use the select list item block to grab a particular item,



4.4 Creating Input Forms and Dynamic Data

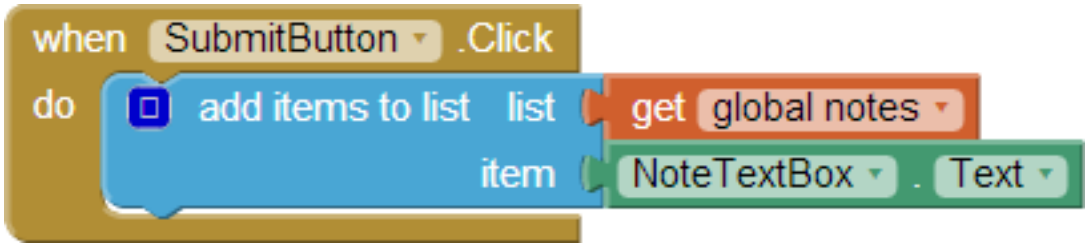
Apps deal with dynamic data: information that changes based on the end user entering new items, or new items being loaded in from a database or web information source.

4.4.1 Defining a Dynamic List

Apps such as a Note Taker begin with an empty list. When you want a list that begins empty, you define it with the create empty list block

4.4.2 Adding an Item

The first time someone launches the app, the notes list is empty. But when the user types some data in a form and taps Submit, new notes will be added to the list.



4.4.3 Removing an Item from a List

You can remove an item from a list by using the remove list item block

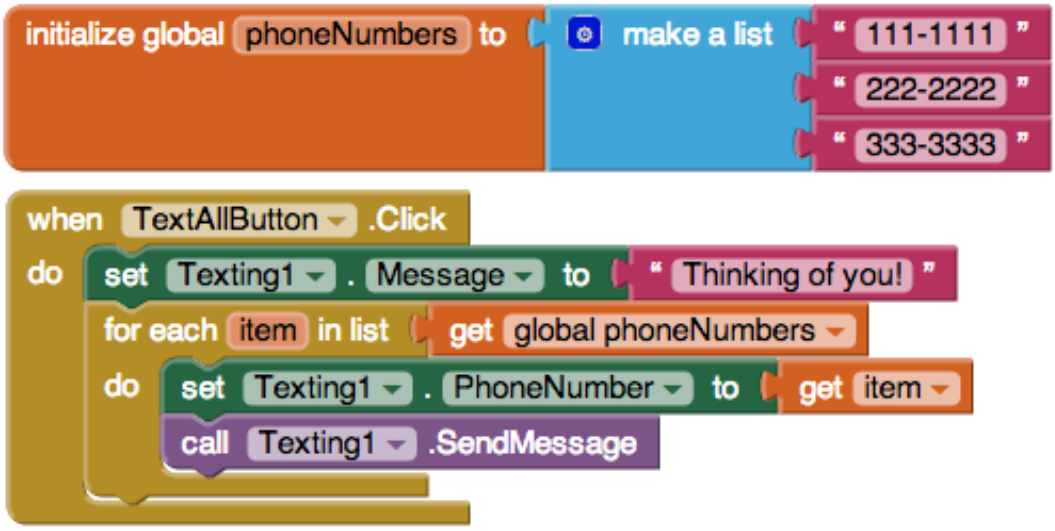
4.5 Lists of Lists

The items of a list can be of any type, including numbers, text, colors, or Boolean values (true/false). But, the items of a list can also, themselves, be lists. Such complex data structures are common.

4.6 Iterating Functions on a List with for each

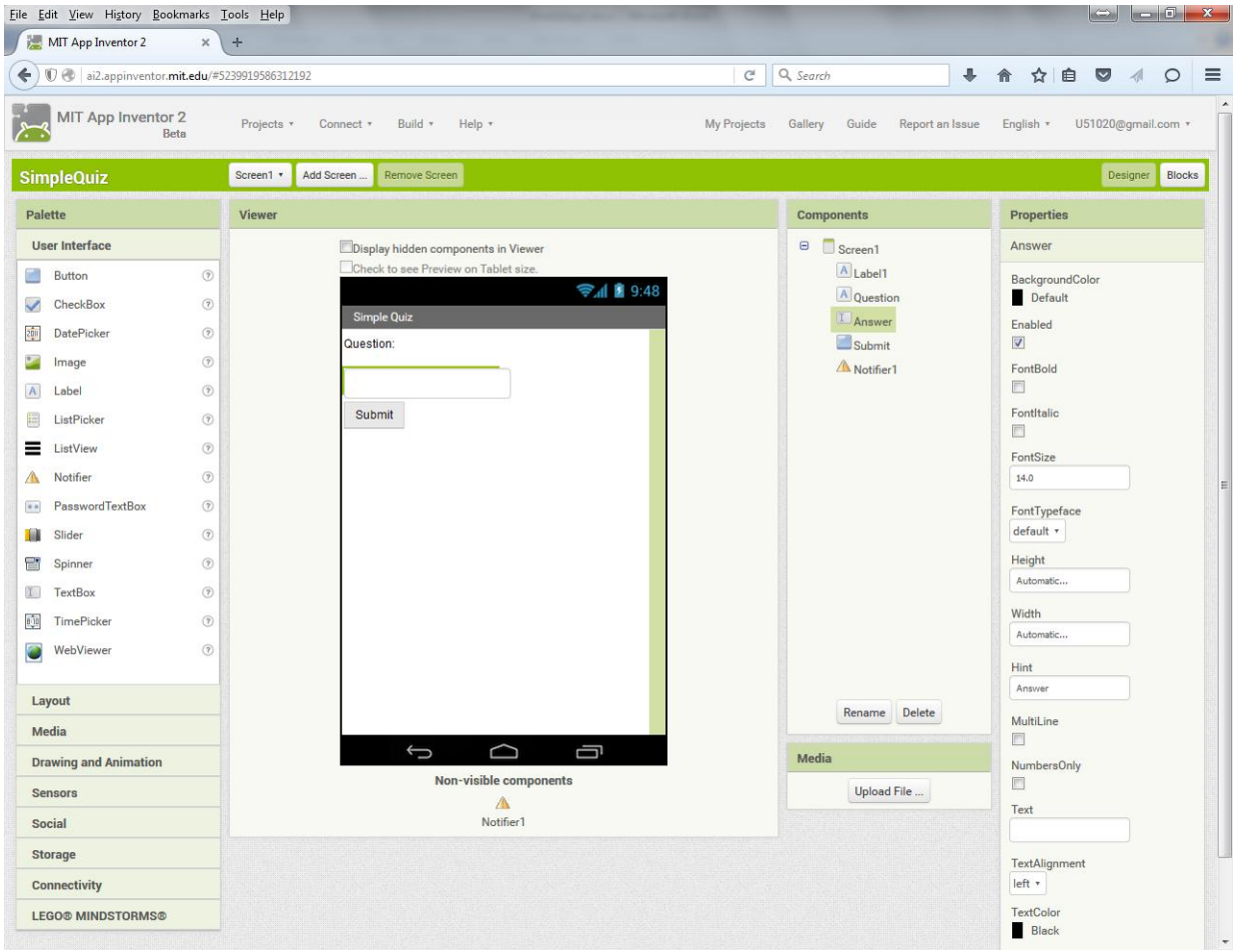
At the top of the for each block, you specify the list that will be processed. The block also has a placeholder variable that comes with the for each. By default, this placeholder is named “item.” You can leave it that way or rename it. This variable represents the current item being processed in the list.

If a list has three items, the inner blocks will be executed three times. The inner blocks are said to be subordinate to, or nested within, the for each block. We say that the program counter "loops" back up when it reaches the bottom block within the for each.



4.7 Exercise: Simple Quiz

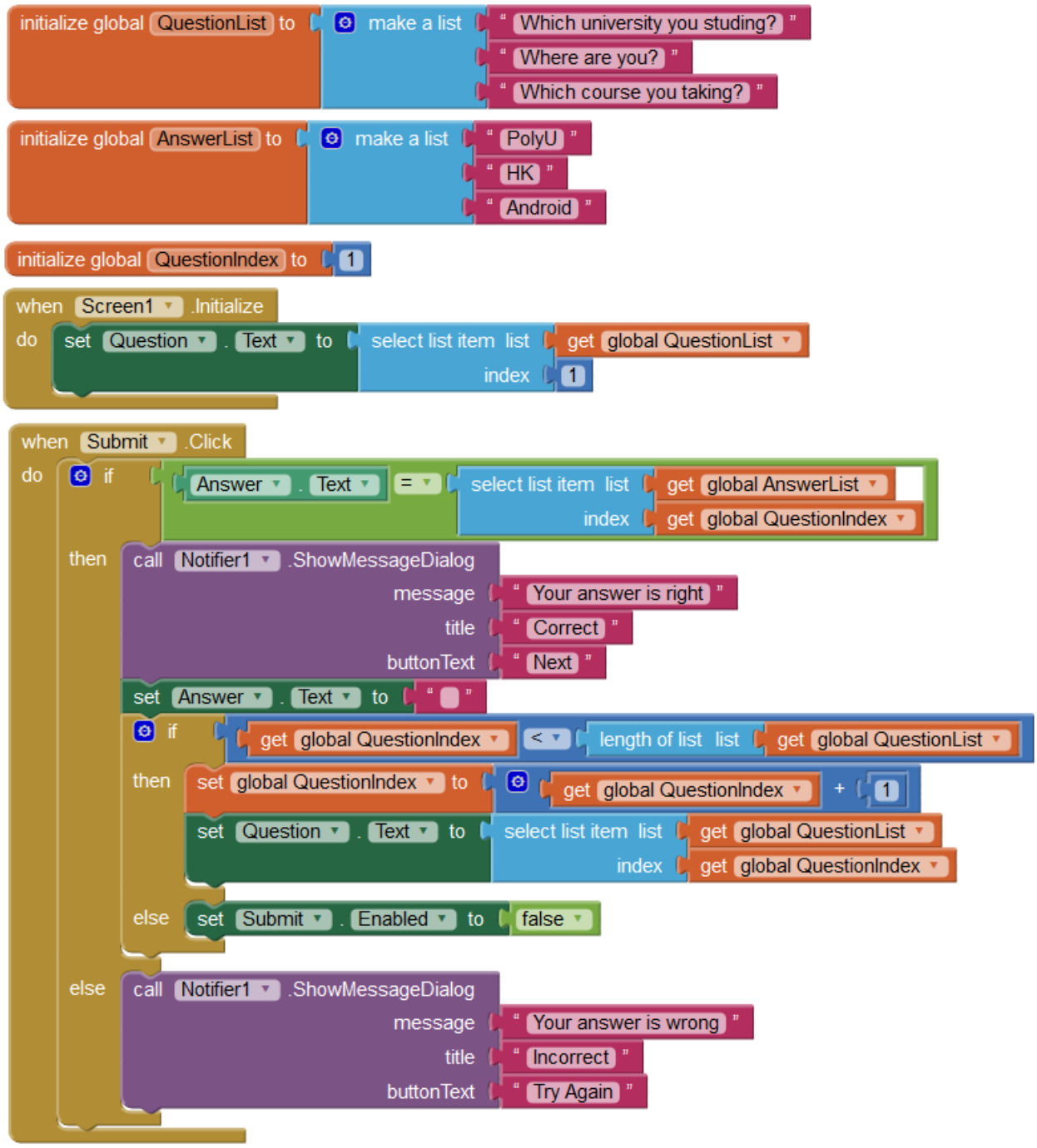
4.7.1 Designer View



4.7.2 Components

Component	Name	Properties	Remark
Screen	Screen1	Title = "**Simple Quiz**"	
Label	Label1	Text = "**Question:**"	
	Question	Text = *[Blank]*	
TextBox	Answer	Hint = "**Answer**"	
Button	Submit	Text = "**Submit**"	
Notifier	Notifier1		

4.7.3 Block Configuration



4.7.4 Sample Output