starting out with >>> APP INVENTOR FOR ANDROID

CHAPTER 4

Decision Blocks and Boolean Logic

TONY GADDIS · REBECCA HALSEY

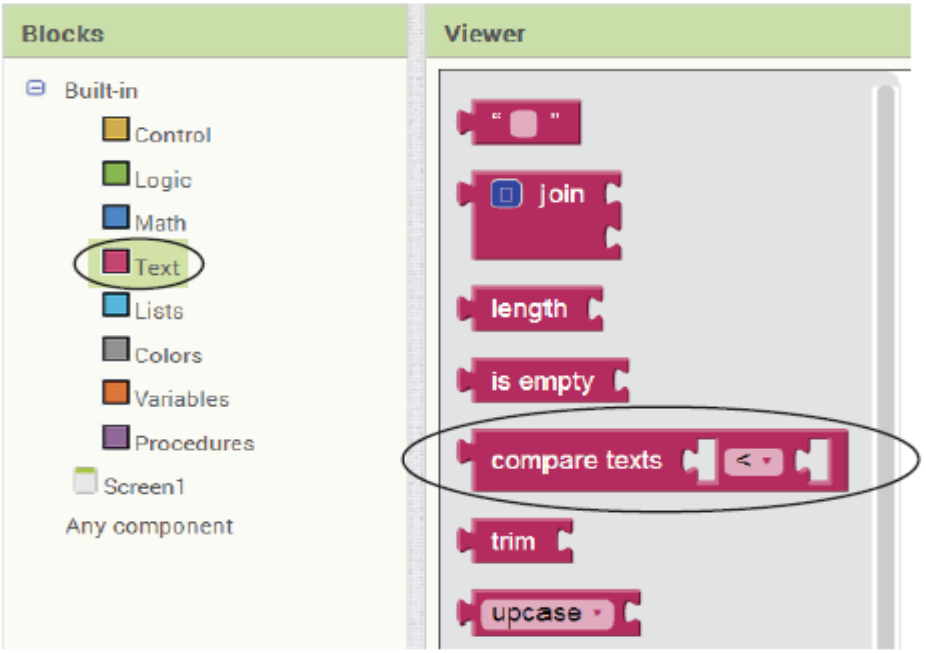
Topics

- Introduction to Decision Blocks
- Relational Operators and the *if* Block
- The *if then else* block
- A First Look at Comparing Strings
- Logical Operators
- Nested Decision Blocks
- The *if then else if* block
- Working with Random Numbers
- The *Screen's Initialize* Event
- The *ListPicker* Component
- The *Checkbox* Component

A First Look At Comparing Strings

- The *compare texts* block compares to strings.
- *compare texts*, found in the *Text* drawer of the *Built-in* section of the *Blocks* column as shown in Figure 4-50.

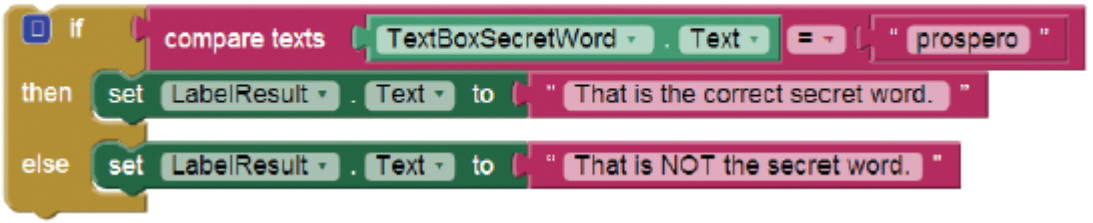
Figure 4-50 The compare texts Block (*Source:* MIT App Inventor 2)



A First Look At Comparing Strings

- The *compare texts* Block let's you compare two strings and determine whether one string is alphabetically less than, greater than, or equal to another string.
- *compare texts* = operator has to sockets. The operator returns true if they are equal. Otherwise it returns false.

Figure 4-52 Comparing Two Strings (*Source:* MIT App Inventor 2)



Logical Operators

- The logical *and* operator and the logical *or* operator allow you to connect multiple Boolean expressions to create a compound expression.
- Logical operators are used to create complex Boolean expressions.

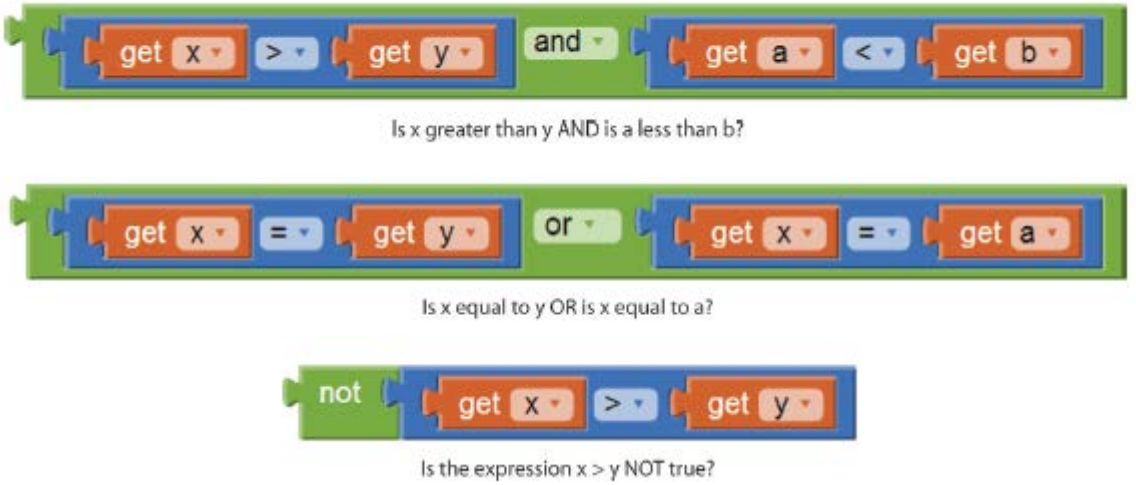
Table 4-4 Logical Operator Blocks (*Source:* Pearson Education, Inc.)

Operator Block	Description
	This is the logical `and` operator. It has sockets for two Boolean expressions. The `and` block returns `true` if both of the Boolean expressions are `true`, or `false` otherwise.
	This is the logical `or` operator. It has sockets for two Boolean expressions. The `or` block returns `true` if either of the Boolean expressions are `true`. If both of the Boolean expressions are `false`, the `or` block returns `false`.
	This is the logical `not` operator. It is a unary operator, meaning it works with only one operand (you can plug only one Boolean expression into this block). The `not` operator reverses the truth of the expression that is plugged into it. If it is applied to an expression that is `true`, the operator returns `false`. If it is applied to an expression that is `false`, the operator returns `true`.

Logical Operators

Figure 4-53 shows some examples of using the logical operator blocks.

Figure 4-53 Logical Operator Block Examples (*Source:* MIT App Inventor 2)



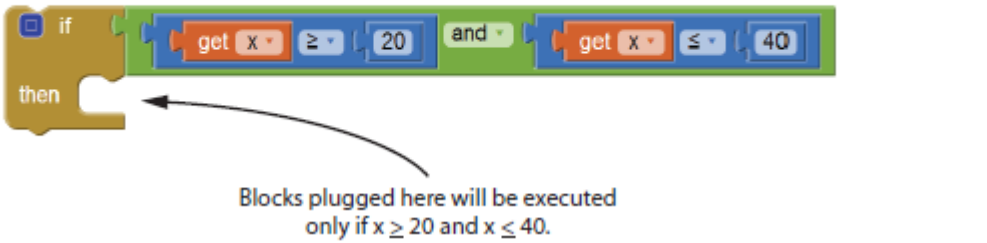
Logical Operators

Checking Numeric Ranges with Logical Operators

- When determining whether a number is inside a range, it is best to use the *and* operator.
- The compound Boolean expression being tested by the `if` block in Figure 4-54 is true only when *x* is greater than or equal to 20 and less than or equal to 40.

Figure 4-54 Determining Whether a Number is Inside a Numeric Range

(*Source:* MIT App Inventor 2)

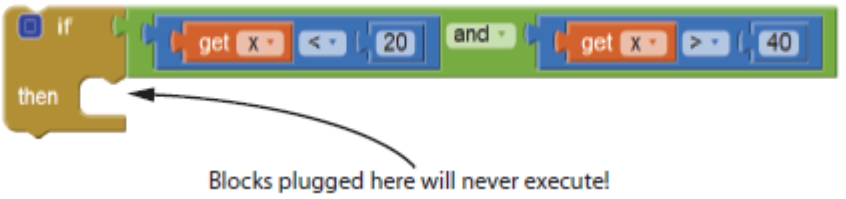


Logical Operators

Checking Numeric Ranges with Logical Operators

- When determining whether a number is outside a range, it is best to use the *or* operator.
- The compound Boolean expression shown in Figure 4-56 would never test true as *x* cannot be less than 20 and at the same time be greater than 40.

Figure 4-56 Logic Error (*Source:* MIT App Inventor 2)



Nested Decision Blocks

A *nested* decision block is written inside the `then` or `else` section of another decision block.

For example in Tutorial 4-5 you will create an app that reads a test score and displays a grade.

The logic of determining the grade can be expressed like this:

If the test score is less than 60, then the grade is F.

Else, if the test score is less than 70, then the grade is D.

Else, if the test score is less than 80, then the grade is C.

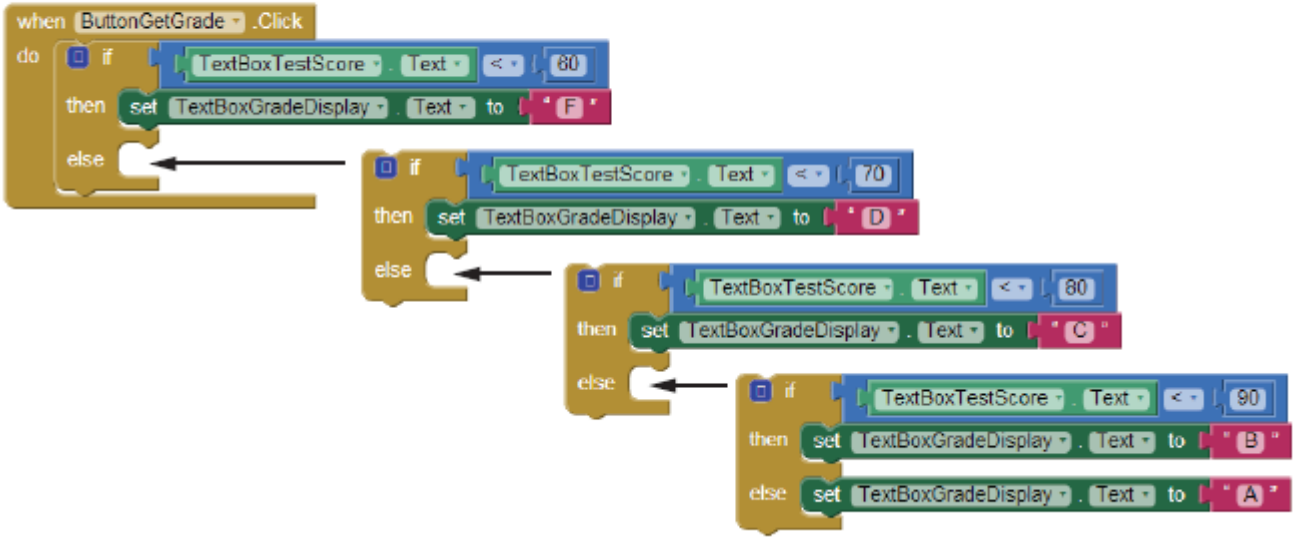
Else, if the test score is less than 90, then the grade is B.

Else, the grade is A .

Nested Decision Blocks

Figure 4-61 Assembling the Nested Decision Blocks in the Grader App

(*Source:* MIT App Inventor 2)



The `if then else if` Block

The *`if then else if block`* tests a series of conditions. It is often simpler to test a series of conditions with the *`if then else if`* block then with a set of nested *`if then else`* blocks.

The *if then else if* Block

- Use the *if then* block's mutator bubble to convert the block into an *if then else* block as in Figure 4-65.
- Click and drag the *else if block* from the left side of the bubble and insert it on the right side of the bubble.
- Drag the *else* block from the left side of the and insert it into the right side of the bubble as in Figure 4-66.

The `if then else if` Block

Figure 4-65 Changing an `if then` Block to an `if then else if` Block
(*Source:* MIT App Inventor 2)

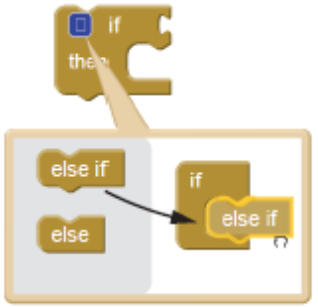
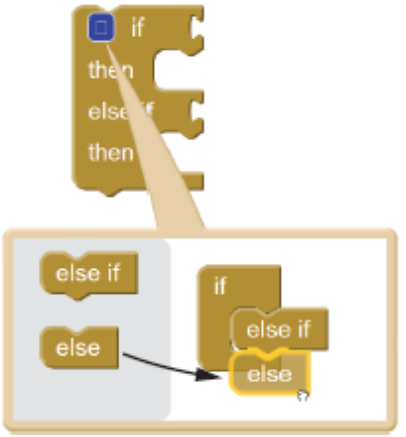


Figure 4-66 Adding an `else` Section to an `if then else if` Block
(*Source:* MIT App Inventor 2)



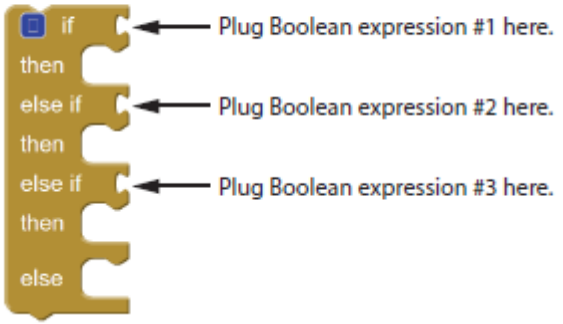
The `if then else if` Block

- When the *`if then else if`* block executes, Boolean expression #1 is tested.
- If the Boolean expression #1 is true, the instructions in the *`then`* socket that immediately follow are executed and the rest of the block is ignored.
- If Boolean expression #1 is false, however, the program jumps to the very next *`else if`* section and tests Boolean expression number two.

The `if then else if` Block

Use the mutator bubble to add as many else if sections as you need as in Figure 4-68.

Figure 4-68 An `if then else if` Block that Can Test Three Boolean Expressions
(*Source:* MIT App Inventor 2)



Working with Random Numbers

Random numbers are useful for lots of different programming tasks. Random numbers are:

- Commonly used in games such as rolling dice or cards.
- In simulation programs. In some simulations, the computer must randomly decide how some living being will behave.
- Useful in statistical programs that must randomly select data for analysis.
- Commonly used in computer security to encrypt sensitive data.

Working with Random Numbers

App Inventor provides blocks for generating random numbers as shown in Figure 4-70. Here is a summary of each block:

- The *random integer* block is a function that takes two arguments: *from* and *to*.
- The *random fraction* block is a function that returns a random fractional number between zero and one.
- The *random set seed* function let's you specify a seed value for random number generation.

Working with Random Numbers

Figure 4-70 The Random Number Blocks (*Source:* MIT App Inventor 2)

Working with Random Numbers

- Figure 4-71 shows the screens for the *RandomNumberDemo* project. When the user clicks the *Generate Random Fraction* button, the app displays a random fraction between 0 and 1.
- When the user clicks the *Generate Random Integer* button, the app displays a random integer within the range of 1 and 100.
- Figure 4-72 shows the *Click* event handlers and Figure 4-73 shows the app running in the emulator.

Working with Random Numbers

Figure 4-72 The `Click` Event Handlers (*Source:* MIT App Inventor 2)

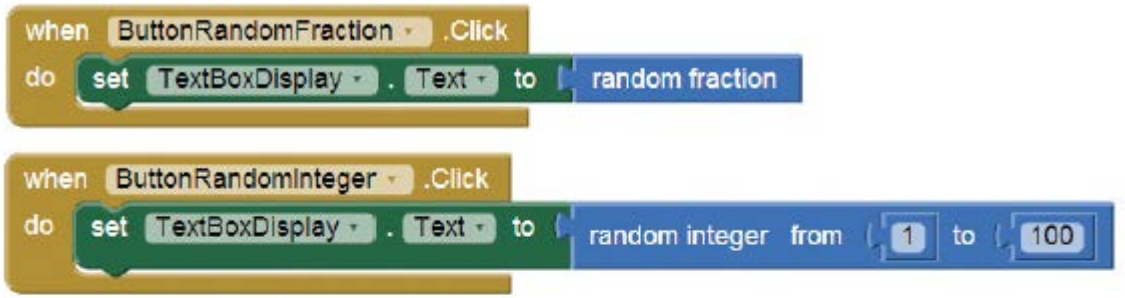


Figure 4-73 The App Running in the Emulator (*Source:* MIT App Inventor 2)