CHAPTER 3

Input, Variables, and Calculations

GLOBAL EDITION
Starting Out with App Inventor for Android
Tony Gaddis • Rebecca Halsey
ALWAYS LEARNING
PEARSON

Topics

- The `TextBox` Component
- Performing Calculations
- Storing Data with Variables
- Creating Blocks with Typeblocking
- The `Slider` Component
- Math Functions

The `TextBox` Component

- The `TextBox` component is a rectangular area that can display text, and can also accept keyboard input.
- In the Designer, the `TextBox` is located in the User Interface section of the Palette.
- `TextBox` components are automatically given default names such as `TextBox1`.
- It is a good idea to change a component's default name to something meaningful.
- When the user types into a `TextBox` component, the text is stored in the component's `Text` property.

The `TextBox` Component

Figure 3-1 Shows a screen from the example project. This is a summary of its components:

- `TableArrangement1` – A `TableArrangement` with one row and two columns.
- `LabelEnterYourName` – A label that displays the text *Enter your name:*.
- `TextBoxName` – A `TextBox` component for the user to enter his or her name.

The TextBox Component

Figure 3-1 Shows a screen from the example project. This is a summary of its components:

- ButtonReadInput – A Button component that, when clicked, reads input that the user typed into the TextBox component, and displays the text in the LabelOutput component.
- TableArrangement2 – A TableArrangement with one row and two columns.

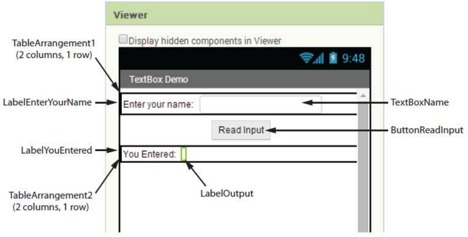
The TextBox Component

Figure 3-1 Shows a screen from the example project. This is a summary of its components:

- LabelYouEntered – A Label that displays the text *You entered*.
- LabelOutput – A component that initially displays nothing when the user clicks the ButtonReadInput component. The text that the user entered into the TextBox name component is displayed in this label.

The TextBox Component

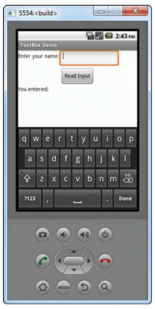
Figure 3-1 Example Project Using a TextBox Component (*Source:* MIT App Inventor 2)



The TextBox Component

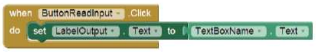
When the user clicks a TextBox, the emulator's virtual keyboard pops up on the screen.

Figure 3-2 The Example App Running in the Emulator (*Source:* MIT App Inventor 2)



The TextBox Component

Figure 3-3 The ButtonReadInput Click Event Handler (Source: MIT App Inventor 2)



when ButtonReadInput .Click
do set LabelOutput . Text to TextBoxName . Text

The the Click event handler for the ButtonReadInput component is shown (Figure 3-3).

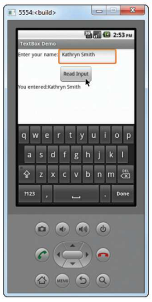
The blocks inside the event handler set the LabelOutput component’s Text property.

PEARSON

The TextBox Component

Figure 3-4 shows the app running in the emulator after the user has entered *Kathryn Smith.*

Figure 3-4 The App after the User has Entered Input and Clicked the Button (Source: MIT App Inventor 2)



PEARSON

The TextBox Component

Other TextBox Properties

- BackgroundColor – Sets the TextBox’s background color.
- Enabled – If checked, the user is able to enter input into the TextBox.
- FontBold, FontItalic, and FontSize – Affect the font of the text displayed in the TextBox.
- Hint – Displays a hint for the user.
- MultiLine – If checked, the TextBox will allow the user to enter multiple lines of input.
- NumbersOnly – If check, TextBox will only allow numbers to be entered.

PEARSON

The TextBox Component

Other TextBox Properties

•TextAignment – Specifies how the text inside the TextBox is aligned. It may be set to *left*, *center*, or *right*.

•TextColor – Sets the color of the text displayed in the TextBox.

•Visible – Specifies whether the component is visible on the screen or hidden.

•Width and Height – Determines the control’s width and height. May be set to *Automatic*, *Fill parent*, or a specific number of pixels.

PEARSON

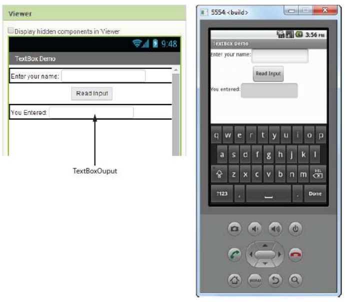
The `TextBox` Component

Using `TextBox` Components to Display Text

- `TextBox` components can also be used to display text.
- In Figure 3-5 the `TextBox` component appears clearly on the screen as a rectangular area.
- Sometimes it is helpful to the user to see the area on the screen where the output will be displayed.
- When using `TextBox` to display text (and not read input), it is a good idea to uncheck the component's `Enabled` property. That prevents the user from selecting it and entering input.

The `TextBox` Component

Figure 3-5 The Modified TextBoxDemo Project (*Source:* MIT App Inventor 2)

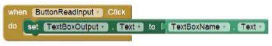


The `TextBox` Component

Using `TextBox` Components to Display Text

- If the `TextBoxDemo` displays its output in a `TextBox` instead of a `Label`, we need to modify the `Click` event handler for the `ButtonReadInput` component.
- Figure 3-6 shows the new event handler.

Figure 3-6 The Modified Click Event Handler for the `ButtonReadInput` Component (*Source:* MIT App Inventor 2)



```
when ButtonReadInput .Click
do  set TextBoxOutput . Text to TextBoxName . Text
```

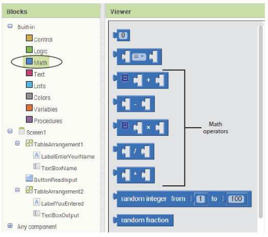
Performing Calculations

- You can use math operators to write expressions that perform simple calculations. The result of a math expression can be assigned to variable.
- A programmer's tool for performing calculations are the *math operators*.

Performing Calculations

In the Block's Editor, you will find the math operators by going into the *Built-in* section, then opening the *Math* drawer. There are four math operator blocks shown in Table 3-1.

Figure 3-8 The Math Operator Blocks (*Source:* MIT App Inventor 2)



Performing Calculations

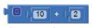
Table 3-1 Math Operator Blocks (*Source:* Pearson Education, Inc.)

Operator	Name of the Operator	Description
	Addition	Adds two numbers and gives the result
	Subtraction	Subtracts one number from another and gives the result
	Multiplication	Multiplies one number by another and gives the result
	Division	Divides one number by another and gives the result
	Exponent	Raises one number to the power of another number and gives the result.

Performing Calculations

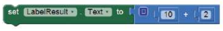
- Each of the operator blocks has its mass symbol displayed in the center with two sockets.
- The two sockets are used to hold *operands*.

Figure 3-9 Using the + Operator Block (*Source:* MIT App Inventor 2)



- We have to plug the + operator block into another block.
- Figure 3-10 shows how we can set the label's `Text` property to the value of the + operator block.

Figure 3-10 Displaying the Result of the + Operator in a Label (*Source:* MIT App Inventor 2)

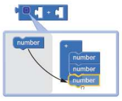


Performing Calculations

Mutator Blocks

- The + and x operator blocks have a blue box (□) in their upper-left corner.
- The block is a *mutator* block.
- Click the blue box (□) that appears in the block's upper-left corner.
- This causes the bubble shown in Figure 3-18 to appear.
- Click and drag the number block (number) from the left side of the bubble.

Figure 3-19 Adding an Additional Operand (*Source:* MIT App Inventor 2)



Performing Calculations

Figure 3-20 The + Block with Three Operands (*Source:* MIT App Inventor 2)

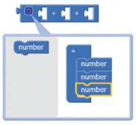
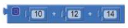


Figure 3-21 The + Block with Three Operands (*Source:* MIT App Inventor 2)

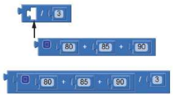


Performing Calculations

Combining Operator Blocks

- You can combine operator blocks to create more complex expressions.
- Figure 3-23 shows how to create the expression by combining a + block with a /

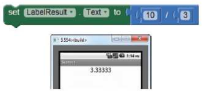
Figure 3-23 Calculating the Average of 80, 85, and 90 (*Source:* MIT App Inventor 2)



Performing Calculations

Formatting Numbers to a Specified Number of Decimal Places

Figure 3-24 Displaying the Result of `10/3` (*Source:* MIT App Inventor 2)

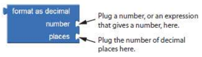


Performing Calculations

Formatting Numbers to a Specified Number of Decimal Places

- You can round a number to a specified number of decimal places using `format as decimal`.

Figure 3-25 The `format as decimal` Block (*Source:* MIT App Inventor 2)

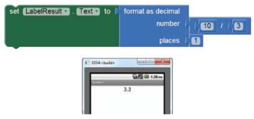

format as decimal
number
places
Plug a number, or an expression that gives a number, here.
Plug the number of decimal places here.

Performing Calculations

Formatting Numbers to a Specified Number of Decimal Places

- The *number* socket requires a `number` or an expression that gives the number. It is the value that you want to round.
- The *places* socket requires the number of decimal places.

Figure 3-26 Rounding the Result of `10/3 to one decimal place`
(*Source:* MIT App Inventor 2)



Performing Calculations

Terminology: Functions, Calling Functions, and Passing Arguments

- A *function* is a method that performs an operation and then *returns* a value.
- When you *execute* a function, we say that you are *calling* it.
- Often functions require additional pieces of data in order for the function to operate.
- When we provide *arguments* to a function, we say that we are *passing the arguments* to the function.

Storing Data with Variables

- A *variable* is a name that represents a value stored in the computer's memory.
- So far, apps that you have created have stored data only in component properties.
- For instance, a component's `Text` property is used to hold data that you want to display.

Storing Data with Variables

Local Variables and Global Variables

- A *local variable* is created inside a method or function, and it can be accessed only by blocks that are also in that method or function.
- A *global variable* is created outside of all methods and functions in the workspace. It can be accessed by any blocks in the workspace, regardless of which method or function they belong to.

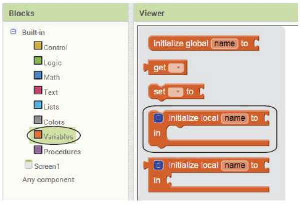
Storing Data with Variables

Creating a Local Variable

- To create a local variable, you must *initialize* it.
- To create and initialize a local variable, open the *Variables* drawer In the *Built-in* section of the Blocks column.
- Notice that in Figure 3-34 there are two blocks that are shaped differently. For now you want to use the one that is circled.

Storing Data with Variables

Figure 3-34 Creating a Variable Initialization Block (*Source:* MIT App Inventor 2)

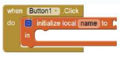


Storing Data with Variables

Creating a Local Variable

When you create an `initialize local` *name* `to` block, place it inside the method or function that it will belong to.

Figure 3-36 An `initialize local` *`name`* `to` Block Placed Inside a Button's `Click` Event Handler (*Source:* MIT App Inventor 2)



The variable initialization block isn't complete yet. We need to:

- Change the variables name to something that describes the variables purpose.
- Assign an initial value to the variable.

Storing Data with Variables

Changing the Variable's Name

The following rules apply to variable names in App Inventor:

- The variable name must begin with an alphabetic letter.
- After the first letter, the remaining characters can be alphabetical letters, numbers, or underscore characters (_).
- You cannot have spaces in a variable name.
- Variable names must be unique within a project.

Storing Data with Variables

Changing the Variable's Name

To change a variable's name, click the word `name` on the `initialize local name to` block.

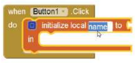


Figure 3-37 Changing the Variable Name (Source: MIT App Inventor 2)

Storing Data with Variables

Assigning an Initial Value to the Variable

- When we set a variable to a value, we are assigning a value to the variable.
- Noticed that the variable initialization block in figure 3-38 has a socket label too.
- This socket requires a value.

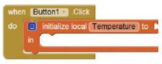


Figure 3-38 The Variable Name Changed to `Temperature` (Source: MIT App Inventor 2)

Storing Data with Variables

Assigning an Initial Value to the Variable

The blocks that you can plug into this socket are:

- `number` blocks
- text string blocks
- Boolean blocks
- List blocks
- `Color` blocks

Storing Data with Variables

Assigning an Initial Value to the Variable

- Figure 3-39 shows two variable initialization blocks. The upper block defines a variable named `Age` and sets its initial value to the number 25.
- The lower block defines a variable named `FirstName` and sets its initial value to the text *Johnny*.

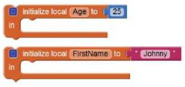

initialize local Age to 25
in
initialize local FirstName to " Johnny "
in

Figure 3-39 Two Complete Variable Initialization Blocks (Source: MIT App Inventor 2)

Storing Data with Variables

Creating a Local Variable That Holds a Number

- Suppose we have a `Click` event handler for a button.
- We want to create a local variable to hold a car's speed.
- We initially assign the number zero to the variable.
- Here are the steps:
  - In the Blocks Editor's *Built-in* section, click *Variables*.
  - Select the initialize `local name to` block as shown in figure 3-40.

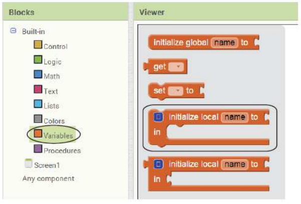
PEARSON

Storing Data with Variables

Creating a Local Variable That Holds a Number

This creates an `initialize local name to` block in your workspace.

Figure 3-40 Creating a Variable Initialization Block (*Source:* MIT App Inventor 2)



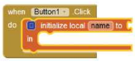
PEARSON

Storing Data with Variables

Creating a Local Variable That Holds a Number

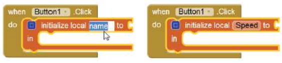
Place the block inside the desired event handler as shown in figure 3-41.

Figure 3-41 Insert the `initialize local` *`name`* `to` Block Inside the Desired Event Handler (*Source:* MIT App Inventor 2)



Click the word *name* and change the name to `Speed` as shown in Figure 3-42.

Figure 3-42 Renaming the Variable (*Source:* MIT App Inventor 2)



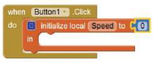
PEARSON

Storing Data with Variables

Creating a Local Variable That Holds a Number

- Create a `number` block to assign to the `Speed` variable.
- In the *Built-in* Section of the Blocks column, click *Math*, then click the `number` block (0).
- Plug the block into the `to` socket of the `Speed` variable initialization block as shown in figure 3-43.

Figure 3-43 Assigning the Number 0 (*Source:* MIT App Inventor 2)



PEARSON

Storing Data with Variables

Creating a Variable That Holds Text

Suppose we have a `Click` event handler for a button and we want to create a variable that holds the text *Dark Roast Coffee*. Here are the steps:

- In the Blocks Editor, go to the *Built-in* section of the Blocks column, click *Variables*.
- Select the `initialize local` *name* `to` block as shown in Figure 3-44.

Storing Data with Variables

Creating a Variable That Holds Text

- This creates an `initialize local` *name* `to` block in your workspace.
- Place the block inside the desired event handler as seen in Figure 3-45.

Figure 3-44 Creating a Variable Initialization Block (*Source:* MIT App Inventor 2)

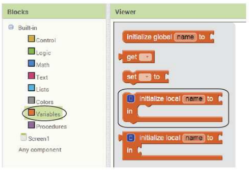
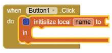


Figure 3-45 Insert the `initialize local` *name* `to` Block Inside the Desired Event Handler (*Source:* MIT App Inventor 2)

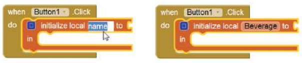


Storing Data with Variables

Creating a Variable That Holds Text

- Change the variable's name to `Beverage`.
- Click the word *name* that appears on the block as in Figure 3-46.

Figure 3-46 Renaming the Variable (*Source:* MIT App Inventor 2)



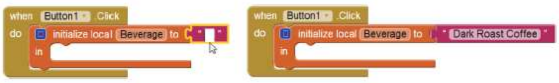
Storing Data with Variables

Creating a Variable That Holds Text

- Create a text string block to assign to the `Beverage` variable.
- In the *Built-in* section of the Blocks column, click `Text`.
- Click the text string block .
- Plug the block into the `to` socket of the `Beverage` variable.
- Click the empty space between the quotation marks, as shown on the left in Figure 3-47.

Storing Data with Variables

Figure 3-47 Assigning the Text *Dark Roast Coffee* (*Source:* MIT App Inventor 2)

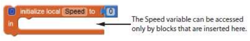

when Button1 .Click
do
initialize local Beverage to " "
in
when Button1 .Click
do
initialize local Beverage to " Dark Roast Coffee "
in

Storing Data with Variables

Working with a Local Variable

The blocks that work with a local variable must be inserted inside the variables initialization block, as shown in figure 3-48.

Figure 3-48 Where to Insert Blocks that Work with a Variable (*Source:* MIT App Inventor 2)

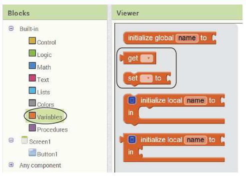

initialize local Speed to 0
in
The Speed variable can be accessed only by blocks that are inserted here.

Storing Data with Variables

Working with a Local Variable

- Use the `get` instruction to get a variables value.
- Use a `set` instruction to store the value in the variable.
- You will find the `get` and `set` blocks in the *Variables* drawer as shown in Figure 3-49.

Figure 3-49 Blocks for Setting and Getting the Value of the `Beverage` Variable (*Source:* MIT App Inventor 2)



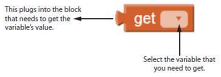
Storing Data with Variables

Working with a Local Variable

When you create a `get` block, you do two things:

1. You plug the `get` block into the block that needs to get the value.
2. On the `get` block, you select the variable that you need to get.

Figure 3-50 Using the `get` Block (*Source:* MIT App Inventor 2)


This plugs into the block that needs to get the variable's value.
get
Select the variable that you need to get.

Storing Data with Variables

Working with a Local Variable

- Figure 3-51 shows that we have created a `get` block and we are going to plug it into the `setLabelFavoriteDrink to` block.
- Next complete the `get` blocked by selecting the `Beverage` variable.
- As shown in figure 3-52 click the down arrow on the `get` block and select `Beverage`.
- Figure 3-53 shows the completed instruction.

Storing Data with Variables

Figure 3-51 Plugging a `get` Block into Another Block (*Source:* MIT App Inventor 2)

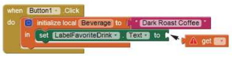

when Button1 .Click
do
initialize local Beverage to " Dark Roast Coffee "
in
set LabelFavoriteDrink . Text to
get

Figure 3-52 Selecting the `Beverage` Variable for the `get` Block (*Source:* MIT App Inventor 2)

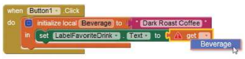
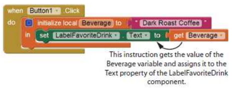

when Button1 .Click
do
initialize local Beverage to " Dark Roast Coffee "
in
set LabelFavoriteDrink . Text to
get
Beverage

Figure 3-53 The Completed Instruction (*Source:* MIT App Inventor 2)


when Button1 .Click
do
initialize local Beverage to " Dark Roast Coffee "
in
set LabelFavoriteDrink . Text to
get Beverage

This instruction gets the value of the Beverage variable and assigns it to the Text property of the LabelFavoriteDrink component.

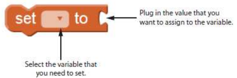
Storing Data with Variables

Working with a Local Variable

When you create a `set` block for a local variable, you do the following things:

- Insert the `set` block into the desired variable's initialization block.
- On the `set` block, select the name of the variable that you want to set.
- Plug a value into the `to` socket of the `set` block.

Figure 3-54 Using the `set` Block (*Source:* MIT App Inventor 2)


set
to
Plug in the value that you want to assign to the variable.
Select the variable that you need to set.

Storing Data with Variables

Working with a Local Variable

Suppose we have a local variable named `Speed`, initialized to the value zero, and we want to change its value to 75. Do the following things:

- Create a `set` block and insert into the speed variables in initialization block as shown in Figure 3-55.
- On the `set` block, select the `Speed` variable as shown in Figure 3-56.
- Create a number block for the value 75 and plug it into the `set` block as shown in Figure 3-57.

Storing Data with Variables

Figure 3-55 The `set` Block Created (*Source:* MIT App Inventor 2)

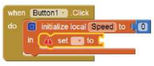


Figure 3-56 Selecting the `Speed` Variable on the `set` Block (*Source:* MIT App Inventor 2)

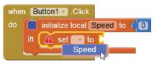
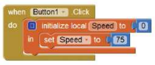


Figure 3-57 Plugging the Value 75 into the `set` Block (*Source:* MIT App Inventor 2)



Storing Data with Variables

Working with a Local Variable

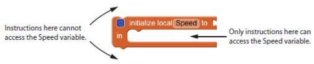
Note: When you create a `set` block, you cannot select the name of the local variable until you plug the set block somewhere inside that local variable's initialization block.

Storing Data with Variables

Variable Scope

- A variable's *scope* is described as part of the program in which a variable may be accessed.
- A variable is visible only to instructions inside the variable's scope.
- The variable can be accessed only by the instructions that are inside the `initialize local` *name* `to` block.
- Figure 3-63 shows an example.

Figure 3-63 The Scope of a Local Variable (*Source:* MIT App Inventor 2)

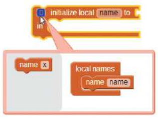

Instructions here cannot access the Speed variable.
initialize local Speed to
in
Only instructions here can access the Speed variable.

Storing Data with Variables

Creating Multiple Local Variables

- The `initialize local` *name* `to` block can be modified to create and initialize multiple variables.
- Click the blue box that appears in the block's upper-left corner to display the mutator bubble as shown in figure 3-64.

Figure 3-64 Mutator Bubble (*Source:* MIT App Inventor 2)

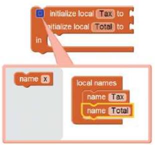


Storing Data with Variables

Creating Multiple Local Variables

- Double click the variable name to change it to something more descriptive.
- Figure 3-67 shows an initialization block that creates two variables named `Tax` and `Total`.

Figure 3-67 The Variable Names Changed to `Tax` and `Total` (*Source:* MIT App Inventor 2)

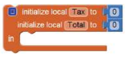


Storing Data with Variables

Creating Multiple Local Variables

The last step is to plug initialization values into each variable as in Figure 3-68.

Figure 3-68 The `Tax` and `Total` Variables Initialized to the Value 0 (*Source:* MIT App Inventor 2)



Storing Data with Variables

Creating Multiple Local Variables

Here is an example that uses to local variables in an event handler.

Figure 3-69 The `SalesTaxCalculator` Project (*Source:* MIT App Inventor 2)

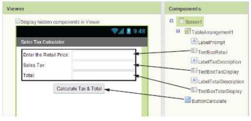
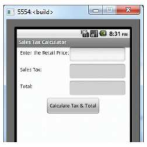


Figure 3-70 The `SalesTaxCalculator` App Running in the Emulator (*Source:* MIT App Inventor 2)

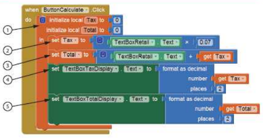


Storing Data with Variables

Creating Multiple Local Variables

The `Click` event handler for the `ButtonReadInput` component is shown in Figure 3-72.

Figure 3-72 `Click` Event Handler for the `ButtonCalculate` Component (*Source:* MIT App Inventor 2)


when ButtonCalculate .Click
do
initialize local Tax to 0
initialize local Total to 0
in
set Tax to TextBoxRetail . Text × 0.07
set Total to TextBoxRetail . Text + get Tax
set TextBoxTaxDisplay . Text to format as decimal number get Tax places 2
set TextBoxTotalDisplay . Text to format as decimal number get Total places 2
1
2
3
4
5

Storing Data with Variables

Figure 3-72 Described

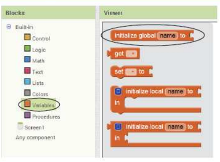
1. This is an initialization block that creates and initializes two local variables: `Tax` and `Total`.
2. This block sets the `Tax` variable to the value of the `TextBoxRetail.Text` x `0.07`.
3. This block sets the `Tax` variable to the value of the `TextBoxRetail.Text` + the value of the `Tax` variable.
4. This block sets the `TextBoxTaxDisplay.Text` to the value of the `Tax` variable, rounded to two decimal places.
5. This block sets the `TextBoxTotalDisplay.Text` to the value of the `Total` variable, rounded to two decimal places.

Storing Data with Variables

Global Variables

- A global variable is created outside of all methods and functions. It is accessible to all of the code in the workspace.
- Create and initialize a global variable by opening the *Variables* drawer found in the *Built-in* section of the Blocks column.

Figure 3-73 Creating a Global Variable Initialization Block (Source: MIT App Inventor 2)

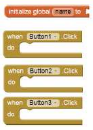


Storing Data with Variables

Global Variables

When you create an `initialize global` *name* `to block`, you can place it anywhere in the workspace that is not inside a method or function.

Figure 3-75 Global Variable Initialization Block Outside of All Methods (Source: MIT App Inventor 2)



Storing Data with Variables

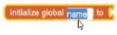
Global Variables

Once you have created a global variable's initialization block, you need to do two more things:

1. Change the variables name to something that describes the variable's purpose.
2. Assign an initial value to the variable.

To change a variable's name, click the word `name` on the `Initialize global` *name* `to` block as shown in Figure 3-76

Figure 3-76 Changing the Name of a Global Variable (Source: MIT App Inventor 2)

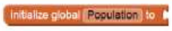


Storing Data with Variables

Global Variables

Figure 3-77 has a socket labeled `to`. This socket requires a value to be plugged in. The value that you plug into this socket is the variable’s initial value.

Figure 3-77 A Global Variable Named `Population` (Source: MIT App Inventor 2)



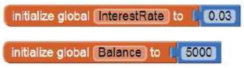
initialize global Population to

Storing Data with Variables

Global Variables

Figure 3-78 shows to global variable initialization blocks.

Figure 3-78 Two Complete Global Variable Initialization Blocks (Source: MIT App Inventor 2)



initialize global InterestRate to 0.03

initialize global Balance to 5000

Once you have created and initialized a global variable, you can use the get block to get the variable’s value and the set block to assign value to the variable.

Storing Data with Variables

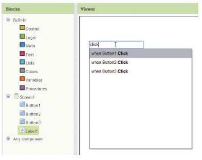
A Word of Caution About Global Variables

- Most programmers agree that you should restrict the use of global variables.
- Here are some reasons:
  - Global variables make debugging difficult.
  - Global variables make a program hard to understand. A global variable can be modified by any instruction in the program.

Creating Blocks with Typeblocking

- *Typeblocking* is a shortcut method for quickly creating blocks using the keyboard.
- In the Blocks Editor click anywhere in the workspace and type part of the name of the block that you want to create.

Figure 3-86 Creating a `Click` Event Handler with Typeblocking (Source: MIT App Inventor 2)



click

when Button1.Click

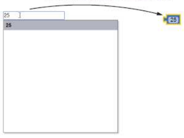
when Button2.Click

when Button3.Click

Creating Blocks with Typeblocking

- Use Typeblocking to quickly create number blocks and text string blocks.
- Suppose you want to create a number block with the value 25.
- Click inside the workspace, type 25, press *Enter*.

Figure 3-87 Creating a Number Block with Typeblocking (Source: MIT App Inventor 2)

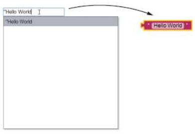


Type 25 and press Enter to create this number block.

Creating Blocks with Typeblocking

- To create a text string block, click inside the workspace, type a quotation mark, type the text you want to set as the block's value, press *Enter*.
- Do not type and ending quotation mark.

Figure 3-88 Creating a Text String Block with Typeblocking (Source: MIT App Inventor 2)



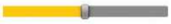
Type "*Hello World* and press Enter to create this text string block.

The Slider Component

The `Slider` component provides a visual way to adjust a value within a range of values.

In the Designer, you will find it in the Basic Pallet.

Figure 3-89 A Slider Component (Source: MIT App Inventor 2)



- The `Slider` component has a `MinValue` property, and a `MaxValue` property that must be set to numeric values.
- By default the `MinValue` property is set to 10.0.
- By default `MaxValue` property is set to 50.0.

The Slider Component

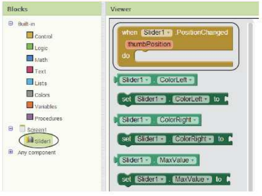
Here is a summary of the Slider components properties:

- `ColorLeft` – Specifies the color of the part of the horizontal track that is to the left of the thumb slider.
- `ColorRight` – Specifies the color of the part of the horizontal track that is to the right of the thumb slider.
- `MaxValue` – The `Slider` component's maximum value.
- `MenValue` – The `Slider` component's minimum value.
- `ThumbPosition` – The position of the thumb slider.
- `Visible` – Determines whether the component is visible on the screen.
- `Width` – The width of the component. It can be set to *Automatic*, *Fill parent*, or a specific number of pixels.

The Slider Component

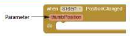
In the Blocks Editor, open the Slider, then select the block for the PositionChanged event handler.

Figure 3-90 Creating a PositionChanged Event Handler (Source: MIT App Inventor 2)



The Slider Component

Figure 3-91 The Slider Component's PositionChanged Event Handler (Source: MIT App Inventor 2)

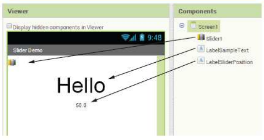


- Figure 3-91 shows and empty PositionChanged event handler.
- thumbPosition is a local variable known as a parameter variable. A parameter variable holds pieces of data passed to an event handler.
- The scope of the thumbPosition parameter variable is the PositionChanged event handler.
- Inside the PositionChanged event handler you can use, get and set blocks.

The Slider Component

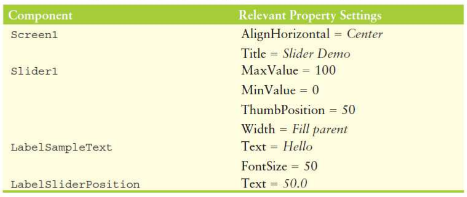
Figure 3-92 shows the screen from the SliderDemo project.

Figure 3-92 The SliderDemo Project (Source: MIT App Inventor 2)



The Slider Component

Table 3-7 Component property settings (Source: Pearson Education, Inc.)



Component	Relevant Property Settings

Screen1	AlignHorizontal = *Center*
	Title = *Slider Demo*

Slider1	MaxValue = 100
	MinValue = 0
	ThumbPosition = 50
	Width = *Fill parent*

LabelSampleText	Text = *Hello*
	FontSize = 50

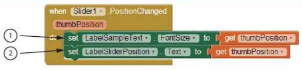
LabelSliderPosition	Text = *50.0*

The Slider Component

Figure 3-95 shows the `Slider1.PositionChanged` event handler.

1. The first set of blocks sets the `LabelSampleText` component's `FontSize` to the value of the `thumbPosition` variable.
2. The second set of blocks sets the `LabelSliderPosition` component's `Text` property to the value of the `thumbPosition` variable.

Figure 3-95 The `Slider1.PositionChanged` Event Handler (*Source:* MIT App Inventor 2)

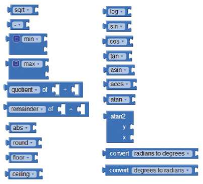


```
when Slider1 .PositionChanged
  thumbPosition
do  set LabelSampleText . FontSize to get thumbPosition
    set LabelSliderPosition . Text to get thumbPosition
```

Math Functions

App Inventor provides numerous advanced math functions for complex calculations.

Figure 3-96 Math Functions (*Source:* MIT App Inventor 2)

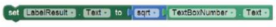


Math Functions

Figure 3-97 shows an example use of the `sqrt` block.

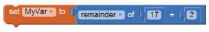
Figure 3-98 shows another example. It sets the variable `MyVar` to the remainder of the 17 divided by 2.

Figure 3-97 Using the `sqrt` Function (*Source:* MIT App Inventor 2)



```
set LabelResult . Text to sqrt TextBoxNumber . Text
```

Figure 3-98 Using the `remainder` Function (*Source:* MIT App Inventor 2)



```
set MyVar to remainder of 17 ÷ 2
```