CHAPTER 6

Procedures and Functions

Topics

- Modularizing Your Code with Procedures
- Passing Arguments to Procedures
- Returning Values from Procedures

Modularizing Your Code with Procedures

- Procedures can be used to break up a complex program into small, manageable pieces.
- Modularization tends to simplify code.
- The benefit of using procedures is known as *code reuse*.

Modularizing Your Code with Procedures

So far you have experienced procedures in the following ways:

•Creating event handlers.

•Executing built-in procedures and functions, such as the `Sound` component's `Play` procedure.

Modularizing Your Code with Procedures

Procedures and Procedures with Results (Functions)

- When you call a procedure, it simply executes the blocks it contains and then terminates.
- When you call a *procedure with results*, it executes the blocks that it contains and then it returns a value back to the block that called it.
- Procedures with the results are also known as *functions*.

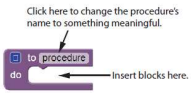
Procedures

- A procedure is a block that contains other blocks.
- In the Block's Editor, go to the *Built-in* section of the Blocks Column, then select *Procedures*.
- In Figure 6-1, select the `to procedure do` block from the drawer. Usually this is referred to as the `procedure` block.

Figure 6-1 The `procedure` Block (Source: MIT App Inventor 2)

Procedures

Figure 6-2 A `procedure` Block (Source: MIT App Inventor 2)



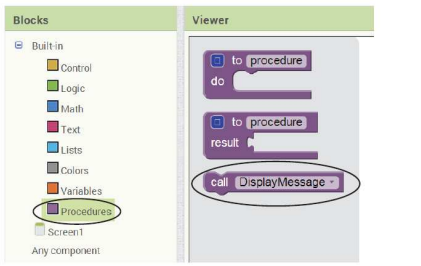
- Figure 6-2 shows and empty procedure block.
- The word that appears at the top of the block is the procedure's default name. Change the name to something more meaningful.

Procedures

- You execute a procedure with a `call` block.
- When you create a `procedure` block, App Inventor automatically creates a `call` block for the procedure, which you will find by opening the Procedures drawer of the Blocks Column.
- Figure 6-4 shows a `call` block for a procedure named `DisplayMessage`.

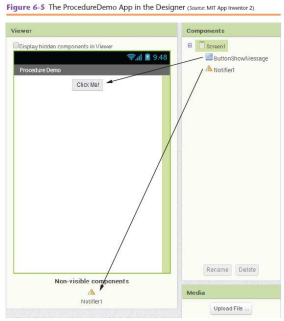
Procedures

Figure 6-4 The `call` Block for a Procedure Named `DisplayMessage` (*Source:* MIT App Inventor 2)



Procedures

- Figure 6-5 shows the ProcedureDemo app in the Designer.
- The app has a `Button` component on its screen and a `Notifier` component for displaying messages.

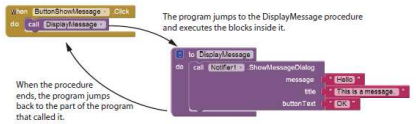


Procedures

- The app has a `Click` event handler for the `ButtonShowMessage` component and a procedure named `DisplayMessage`.
- Inside the `ButtonShowMessage.Click` event handler, we have a `call` block that calls the `DisplayMessage` procedure.
- When the procedure ends as shown in Figure 6-7, the program jumps back to the part of the program that called the `DisplayMessage` procedure.

Procedures

Figure 6-7 Calling a Procedure (*Source:* MIT App Inventor 2)



Procedures

Top-Down Design

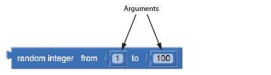
Programmers commonly use a technique known as *top-down design* to break down a program into procedures. The process of top-down design is performed in the following manner:

- The overall task that the program is to perform is broken down into a series of subtasks.
- Each of these subtasks is examined to determine whether it can be further broken down into more subtasks.
- Once all of these subtasks have been identified, they are written in code.

Passing Arguments to Procedures

- Sometimes it is useful to send one or more pieces of data to the procedure. These pieces are known as *arguments*.
- Figure 6-18 shows two arguments. The arguments specify the minimum and maximum values for a random integer.

Figure 6-18 Arguments Passed to the `random integer` Function (Source: MIT App Inventor 2)



Passing Arguments to Procedures

- To equip a procedure block with a parameter value, open the procedures mutator bubble ( ).
- Click and drag the input block from the left side of the bubble and insert it into the right side of the bubble.
- Parameters are variables.
- Variables should have meaningful names.

Passing Arguments to Procedures

Figure 6-19 Adding a Parameter to a Procedure (Source: MIT App Inventor 2)

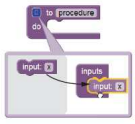
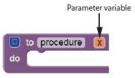


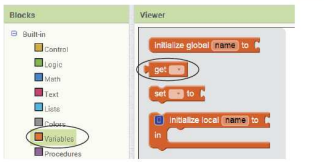
Figure 6-20 A Procedure with a Parameter Named x (Source: MIT App Inventor 2)



Passing Arguments to Procedures

- A parameter's scope is limited to the procedure that it belongs to. To get the value you use a `get` block.
- Use the *Variables* drawer of the Blocks column to create a `get` block as shown in figure 6-22.

Figure 6-22 Selecting the get Block from the *Variables* Drawer (Source: MIT App Inventor 2)



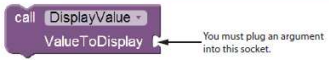
Passing Arguments to Procedures

- When a procedure has a parameter, you must pass an argument to that parameter.
- For example, if an app has the `DisplayValue` procedure as shown in Figure 6-24, the procedure's `call` block will have a socket named `ValueToDisplay` as shown in Figure 6-25.
- When you call the procedure, you must plug an argument into the socket.

Passing Arguments to Procedures

Figure 6-24 Getting the Value of a Parameter (Source: MIT App Inventor 2)

Figure 6-25 Socket for an Argument



Passing Arguments to Procedures

- The app's screen has three buttons, a `Click` event handler has been written for each.
- The workspace has the `DisplayValue` procedure.
- If the user clicks the *Display 5* button, the `Button Display5.Click` event handler executes.
- This causes the value 5 to appear in the text box as shown in Figure 6-29.

Passing Arguments to Procedures

Figure 6-27 The ArgumentDemo App in the Designer (Source: MIT App Inventor 2)

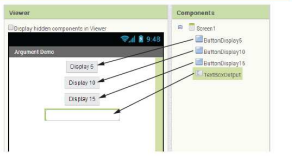
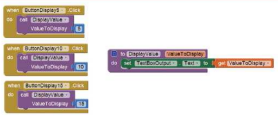
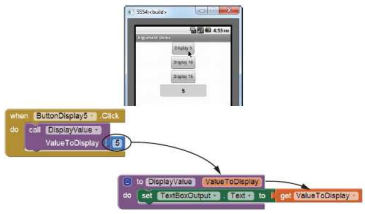


Figure 6-28 The ArgumentDemo App Workspace (Source: MIT App Inventor 2)



Passing Arguments to Procedures

Figure 6-29 Passing the Number 5 as an Argument (Source: MIT App Inventor 2)



Returning Values from Procedures

A procedure with a result, or function, is like a regular procedure in the following ways:

- It contains a group of statements that perform a specific task.
- When you want to execute the function you call it.
- The value that is returned from a procedure can be used like any other value.

Returning Values from Procedures

- The `random integer` function as shown in Figure 6-39, returns a value.
- To use the value that is returned, you plug it into another block.

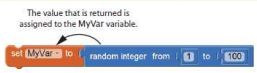
Figure 6-39 The `random integer` Function Returns a Value (Source: MIT App Inventor 2)

Returning Values from Procedures

Figure 6-40 shows the `random integer` function plugged into a variable's set block.

The value that is returned from the `random integer` function is assigned to that variable.

Figure 6-40 Assigning the Returned Value to a Variable (Source: MIT App Inventor 2)



Returning Values from Procedures

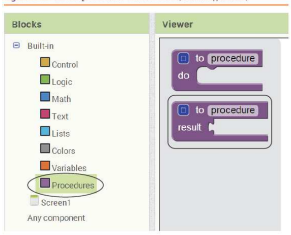
In App Inventor, you create a function the same way you create a regular procedure, with two exceptions:

- You use the `to procedure result` block instead of the `to procedure do` block.
- You must plug a value into the block's `result` socket. This is the value that is returned from the procedure.

Returning Values from Procedures

The `to procedure result` block is in the *Procedures* drawer.

Figure 6-41 The `to procedure result` Block (Source: MIT App Inventor 2)



Returning Values from Procedures

- Figure 6-43 shows an example function. The function's name is `Add` and its purpose is to add two numbers.
- The `Add` function shown in Figure 6-43 is only for demonstration purposes. It isn't necessary to write a function for adding numbers.

Figure 6-43 An Example Function (Source: MIT App Inventor 2)

Returning Values from Procedures

- The app lets you enter your age into the `TextBoxAge1` and your best friend's age into `TextBoxAge2`.
- Click the *Calculate Combined Age* button, and the app displays the sum of the two ages in `TextBoxCombinedAge Display`.

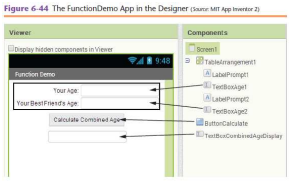


Figure 6-44 The FunctionDemo App in the Designer (Source: MIT App Inventor 2)

Returning Values from Procedures

Here is the app running in the emulator.

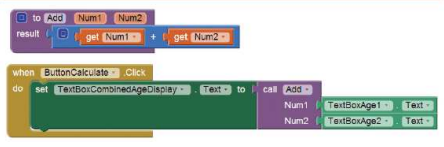
Figure 6-45 The FunctionDemo App Running in the Emulator (Source: MIT App Inventor 2)

Returning Values from Procedures

- Figure 6-46 shows the app's workspace.
- At the top of the workspace is the `Add` function.
- Below that is the `Click` event handler for the `ButtonCalculate` component.
- In the event handler we set the `TextBoxCombinedAge` component's `Text` property to the value that is returned from the `Add` Function.
- The arguments that are passed to the `Add` function are the `Text` properties of the `TextBoxAge1` and `TextBoxAge2` components.

Returning Values from Procedures

Figure 6-46 The FunctionDemo App's Workspace in the Blocks Editor (Source: MIT App Inventor 2)



Returning Values from Procedures

Figure 6-47 shows how these are passed as arguments to the `Add` function.

Figure 6-47 Arguments Passed to the `Add` Function and a Value Returned
(Source: MIT App Inventor 2)