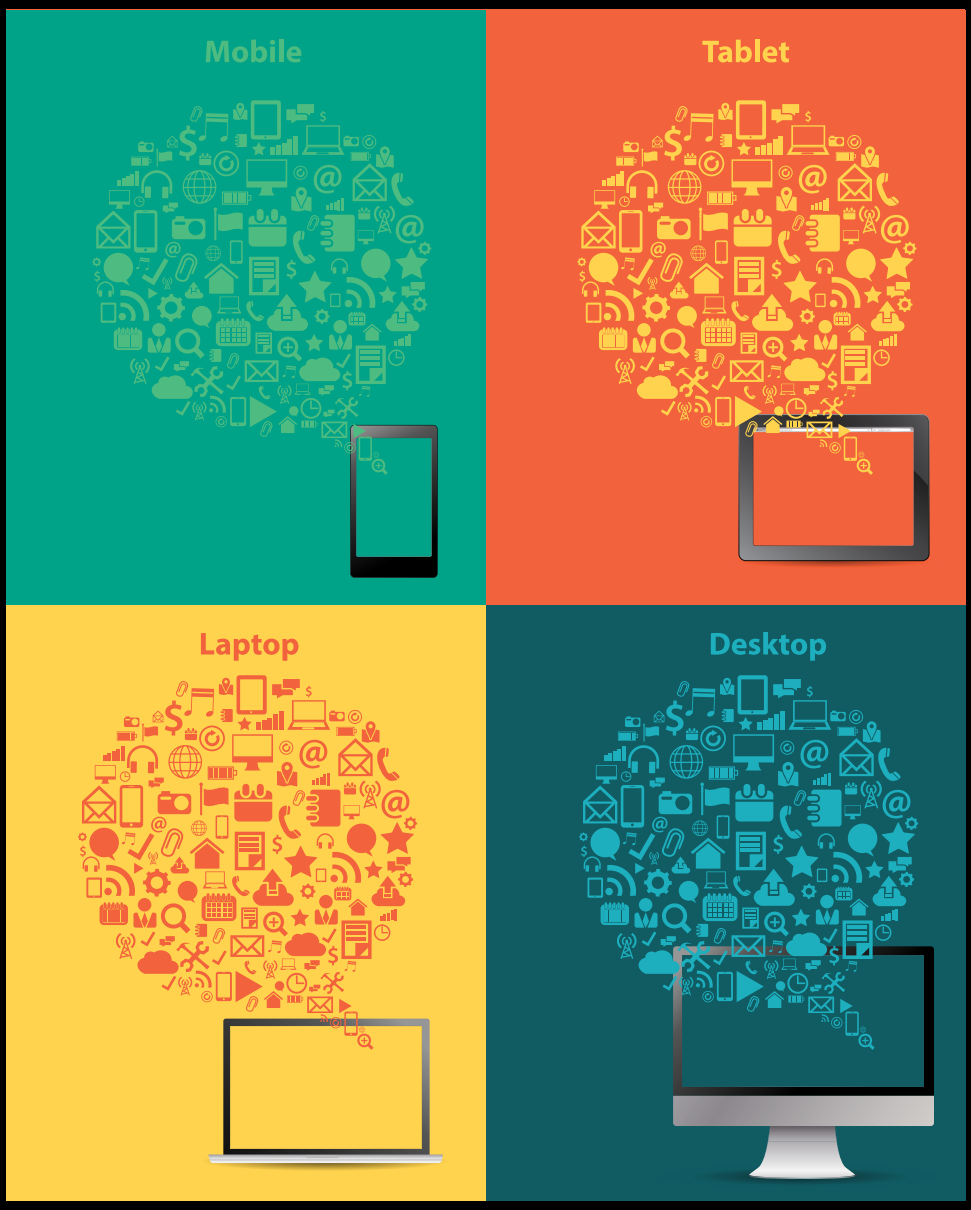
GLOBAL EDITION



Starting Out with App Inventor for Android

Tony Gaddis • Rebecca Halsey

ALWAYS LEARNING

PEARSON

STARTING OUT WITH

First Edition Global Edition

App Inventor for Android

This page intentionally left blank.

STARTING OUT WITH

First Edition Global Edition

App Inventor for Android

Tony Gaddis
and
Rebecca Halsey

PEARSON

Boston Columbus Indianapolis New York San Francisco
Upper Saddle River Amsterdam Cape Town Dubai London Madrid Milan
Munich Paris Montréal Toronto Delhi Mexico City São Paulo Sydney
Hong Kong Seoul Singapore Taipei Tokyo

Vice President and Editorial Director, ECS: *Marcia J. Horton*
Acquisitions Editor: *Matt Goldstein*
Editorial Assistant: *Kelsey Loanes*
Program Manager: *Carole Snyder*
Project Manager: *Rose Kernan, RPK Editorial Services, Inc.*
Project and Program Manager Team Lead: *Scott Disanno*
Media Team: *Steve Wright*
R&P Project Manager: *Rachel Youdelman*
Publishing Administrator and Business Analyst, Global Edition: *Shokhi Shah Khandelwal*
Assistant Acquisitions Editor, Global Edition: *Aditee Agarwal*
Assitant Project Editor, Global Edition: *Sinjita Basu*
Senior Manufacturing Controller, Production, Global Edition: *Trudy Kimber*
Operations Specialist: *Vincent Scelta*
Full-Service Project Management: *iEnergizer Aptara®, Ltd.*
Cover Design: *Lumina Datamatics*
Cover Photo: *Shutterstock/My Life Graphic*
Cover Printer: *Ashford Colour Press*

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsonglobaleditions.com

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

ISBN-13: 978-1-292-08032-1
ISBN-10: 1-292-08032-9

Typeset in Sabon LT Std by iEnergizer Aptara®, Ltd.

Printed and bound by Ashford Colour Press in the United Kingdom.

Brief Contents

Contents

Preface

Cell phones have become an important part of most students' lives. Even students with limited computer experience have no trouble using their phones to send text messages, check their email, and update their Facebook statuses. Of course, the typical cell phone today is much more than a mere phone. It's a powerful computer with many unique capabilities, including the ability to run thousands of available programs, or apps.

Even though students regularly download, install, and use apps on their phones, they do not typically think of their phones as computers. In fact, students have a unique relationship with their phones that is different, and more personal, than the relationship they have with their laptop computers. When students learn that they can create their own mobile apps—especially apps that take advantage of a phone's unique capabilities (such as text messaging, location sensing, etc.)—they become excited and motivated to learn.

This book capitalizes on that excitement and motivation by using App Inventor 2 to teach introductory programming skills. App Inventor 2 is a free, cloud-based development platform that is provided by The MIT Center for Mobile Learning. It allows users with no prior programming experience to make their own Android apps. It is extremely easy to use, and it combines a visual GUI designer with a drag-and-drop code editor. An on-screen Android emulator or an actual Android device that is connected to the computer (either wirelessly or with a USB cable) runs apps as they are created. Because App Inventor 2 allows students to create apps and see them running on a phone, programming becomes a personally meaningful skill.

Programming With Blocks

For many beginning students, learning the syntax of a programming language can be a daunting task. Precious time that should be devoted to learning the fundamentals of programming is often spent tracking down missing semicolons or unbalanced braces.

Syntax errors in App Inventor are never a problem, because they never happen! You build an app by dragging and dropping "blocks" into an editor. The blocks, which represent actions and data, can be snapped together, like the pieces of a puzzle, to create fully functional programming statements. Because you don't have to spend time locating and fixing syntax errors, you can concentrate on planning the actions that you want your app to perform and arranging them into the proper sequence.

Runtime and logic errors can still occur, of course, because the student can use the wrong instruction or get instructions out of order. But because syntax is not an issue, the student devotes his or her time to developing and debugging algorithms.

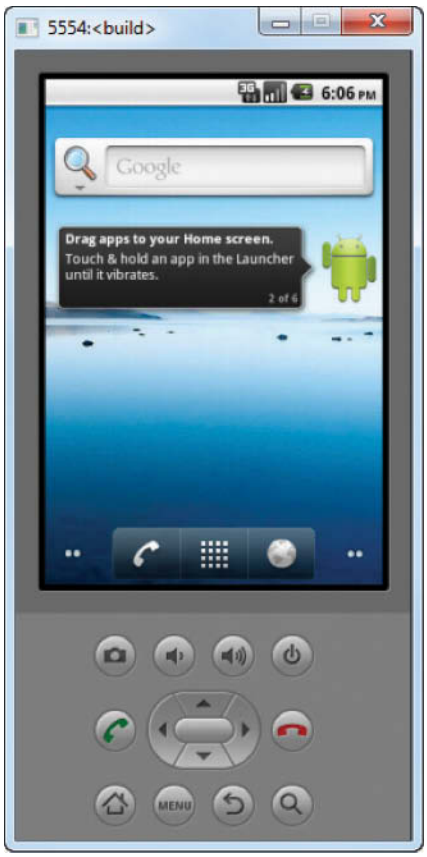
Using the Emulator or Android Devices

You use a Windows, Mac, or Linux computer to develop apps with App Inventor 2, but to test your apps, you use either the Android emulator, which is included with App Inventor, or an actual Android device such as a smartphone or a tablet. An Android device can be connected to the computer either wirelessly (via Wi-Fi) or with a USB cable. This book can be used with either approach.

The emulator, which is shown in Figure P-1, is a simulated Android phone. As you are using App Inventor to develop an app, the app appears and runs on the emulator's screen. You can interact with the emulator in many of the same ways that you interact with an actual smartphone. Although the emulator is limited (for example, it does not have a GPS sensor to report its location, and it cannot make phone calls), it does provide many of the basic features of an actual smartphone.

Most of the topics that are covered in Chapters 1 through 11 can be taught using the emulator. The topics covered in Chapters 12 and 13 require an Android device.

Figure P-1 The Android Emulator



App Inventor in the Classroom

App Inventor can be used in a variety of ways in the classroom, and this text is designed to accommodate all of them. Here are some examples:

- You can use this text with App Inventor 2 for the first part of an introductory programming course, and then switch to a traditional programming language. Depending on the amount of time you want to devote to App Inventor, you can use the entire book, or you can omit some of the latter chapters.
- You can use this text with App Inventor 2 for a brief introduction to programming in a computer concepts course or an introduction to technology course. The latter chapters can be omitted to fit the amount of time that you have.
- You can use this text by itself in a semester-long course that uses only App Inventor 2 to teach programming fundamentals.
- You can use this text in short courses or summer programs that use App Inventor 2 to teach programming.

VideoNotes to Accompany This Book

A full set of VideoNotes has been developed to accompany each tutorial in the book. Students can follow along with the authors as they work through tutorials in the videos. Also, one exercise or programming project at the end of each chapter has an accompanying VideoNote that shows the student how to create the solution. To access these supplements, go to www.pearsonglobaleditions.com/Gaddis and click on the image of this book's cover.

Brief Overview of Each Chapter

Chapter 1: Introduction Programming and App Inventor 2

This chapter explains what algorithms and programs are, and why we use programming languages. App Inventor 2 is introduced and the student learns the fundamental steps for creating an app's user interface, using the Blocks Editor to program the app, and using the emulator to test an app.

Chapter 2: Working With Media

In this chapter, the student learns to create apps that use images and sound. Topics include setting the background image for the device's screen and displaying images in image components, as well as on buttons (to create clickable images). The Sound component is introduced for playing sound effects, and techniques for working with colors are presented. The chapter discusses the visual arrangement of components in the app's user interface and the importance of commenting code.

Chapter 3: Input, Variables, and Calculations

In this chapter, the student learns to use TextBox components to read user input. Variables are introduced as a way to store data in memory. App Inventor's math

operator blocks are introduced, and the student learns to create math expressions. The Slider component is also discussed.

Chapter 4: Decision Blocks and Boolean Logic

In this chapter, the student learns about App Inventor's decision structures: the `if then` block, the `if then else` block, and the `if then else if` block. The relational operators are introduced, as well as logical operators. The chapter discusses random numbers, their applications, and how to generate them in App Inventor. The Screen component's `Initialize` event is introduced. The chapter concludes with a discussion of the ListPicker and CheckBox components.

Chapter 5: Repetition Blocks, Times, and Dates

This chapter shows the student how to use loops to create repetition structures. App Inventor's `while` and `for each` loops are presented. Counters, accumulators, and running totals are also discussed. The chapter introduces the Clock component as a way to work with dates and times, and also as a way to create a timer. The chapter concludes with a discussion of the DatePicker component.

Chapter 6: Procedures and Functions

In this chapter, the student first learns how to write procedures. The chapter shows the benefits of using procedures to modularize programs and discusses the top-down design approach. Then, the student learns to pass arguments to procedures. Finally, the student learns to write functions, or procedures that return a result.

Chapter 7: Lists

This chapter introduces lists. The student learns to create lists, insert and append items, select items at specific and random positions, remove items, replace items, search for items, and more.

Chapter 8: Storing Data on the Device

This chapter discusses the File component and the TinyDB component. The File component allows you to read and write text files on the device or emulator. TinyDB is a simple database component that allows you to store data as tag-value pairs.

Chapter 9: Graphics and Animation

App Inventor provides components for creating graphics and animations. In this chapter, the student first learns to draw primitive graphics with the Canvas component. Then, the Ball and ImageSprite components are discussed. Simple games are created that use collision detection, the Clock component, and sprites.

Chapter 10: Working with Text

In this chapter, the student learns to process strings at a detailed level. Various text-processing capabilities are discussed, such as concatenation, comparing strings, trimming strings, converting case, finding, replacing, and extracting substrings, and string splitting.

Chapter 11: Text to Speech and Text Messaging

This chapter begins with an introduction to the TextToSpeech component, which converts text to spoken words. (The component reads text aloud.) Next, the student learns to use the Texting component to send and receive text messages.

Chapter 12: Sensors

This chapter focuses on the sensors that are found on an Android device. The sensors that are introduced are: The LocationSensor, for determining the device's physical location, the OrientationSensor, for determining the device's orientation in 3D space, and the AccelerometerSensor, for determining the device's acceleration in 3D space. This chapter concludes with a discussion of using the ActivityStarter component to launch Google Maps.

Chapter 13: Other App Inventor Capabilities

This chapter presents various components that work on Android devices. The components that are covered in this chapter give capabilities such as recording audio, taking photos, selecting images from the device's gallery, playing videos, selecting entries from the contact list, scanning barcodes, using voice recognition, connecting to a Twitter account, and storing data on a Web server with a TinyWebDB component.

Appendix A: Setting Up App Inventor

Appendix B: Connecting an Android Device to App Inventor

Appendix C: Uploading Your Application to App Inventor Gallery and Google Play Store

Appendix D: Component Reference

Appendix E: Answers to Checkpoints

Features of the Text

Concept Statements

The major sections of the text starts with a concept statement. This statement concisely summarizes the main point of the section.

Example Apps

The text has an abundant number of complete and partial example apps, which are each designed to highlight the topic currently being studied.

Tutorials

Each chapter has several hands-on tutorials that lead the student through the process of developing or completing an app. These tutorials give the student experience performing the tasks discussed in the chapters.

VideoNotes

Online videos developed specifically for this book are available for viewing at www.pearsonglobaleditions.com/Gaddis. Icons appear throughout the text, alerting the student to videos about specific topics.

Notes

Notes appear at several places throughout the text. They are short explanations of interesting or frequently misunderstood points relevant to the topic at hand.

Tips

Tips advise the student on the best techniques for approaching different programming problems.

Checkpoints

Checkpoints are questions placed at intervals throughout each chapter. They are designed to query the student's knowledge quickly after learning a new topic.

Review Questions

Each chapter presents a thorough set of multiple-choice and short-answer review questions.

Exercises

Each chapter offers a set of exercises for developing apps. The exercises are designed to solidify the student's knowledge of the topics presented in the chapter.

Online Resources

This book's online resource page contains numerous student supplements. To access these supplements, go to www.pearsonglobaleditions.com/Gaddis and click on the image of this book's cover. You will find the following items:

- A link to the App Inventor site
- The book's example apps
- Graphics and audio files that can be used in student projects
- Access to the book's companion VideoNotes

Instructor Resources

The following supplements are available to qualified instructors only:

- Answers to the Review Questions
- Solutions for the exercises
- PowerPoint presentation slides for each chapter

Visit the Pearson Instructor Resource Center (www.pearsonglobaleditions.com/Gaddis) or send an e-mail to computing@pearson.com for information on how to access them.

Acknowledgements

The authors would like to thank Dr. Hal Abelson of MIT for his inspiring work, and particularly for creating App Inventor. We want to thank the entire App Inventor team at MIT for the amazing job they are doing. We also want to thank everyone at Pearson Education for making this book possible. We are extremely grateful that Matt Goldstein is our editor. He and Kelsey Loanes, editorial assistant, guided us through the process of writing this book. We are also fortunate to have Demetrius Hall and Bram Van Kempen as marketing managers. They do an amazing job of getting computer science books out to the academic community. The production team, lead by Camille Trentacoste, worked tirelessly to make this book a reality. We could not have done it without their patience and hard work. Thanks to you all!

Pearson wishes to thank and acknowledge the following people for their work on the Global Edition:

Contributor:
Passent M. El-Kafrawy, *Menoufia University, Egypt*

Reviewers:
Muthuraj M., *Android Developer, Bangalore*
Arup Bhattacharjee, *RCC Institute of Technology, India*
Soumen Mukherjee, *RCC Institute of Technology, India*
Manasa Rangarer, *NMAM Institute of Technology, Nitte, India*
Professor Harsh Bhasin, *Jamia Hamdard*

This page intentionally left blank.

About the Authors

Tony Gaddis

Tony Gaddis is the author of the *Starting Out with* series of textbooks. Tony has nearly twenty years of experience teaching computer science courses, primarily at Haywood Community College. He is a highly acclaimed instructor who was previously selected as the North Carolina Community College "Teacher of the Year" and has received the Teaching Excellence award from the National Institute for Staff and Organizational Development. The *Starting Out with* series includes introductory books covering C++, Java™, Microsoft® Visual Basic®, Microsoft® C#®, Python, Programming Logic and Design, and Alice, all published by Pearson Education.

Rebecca Halsey

Rebecca Halsey is an Associate Professor at Guilford Technical Community College where she teaches classes in Computer Science and Mobile Application Development. She is also developing and leading the new Mobile Application Development curriculum at GTCC. She also has twenty years of industry experience as a software developer.

This page intentionally left blank.

VideoNotes

CHAPTER 7

CHAPTER 8

CHAPTER 9

CHAPTER 10

CHAPTER 11

CHAPTER 12

CHAPTER 13

CHAPTER 1

Introduction to Programming and App Inventor

TOPICS

1.1 Introduction

This book teaches fundamental programming skills using an exciting application known as App Inventor 2. (We will refer to it simply as App Inventor.) App Inventor allows you to quickly and easily create applications, or "apps," for Android smartphones and tablets. It is not necessary to have prior programming experience or knowledge to use this book. App Inventor was created for beginners who have never programmed before.

You might find it surprising that with no previous programming experience, you can learn to create apps for a smartphone or a tablet. Perhaps you have heard that you need to know a lot about programming in languages such as Java to create mobile apps. While it is true that apps are typically created with high-level programming languages, App Inventor takes a different approach. With App Inventor, you use a screen designer to visually create an app's screen, as shown in Figure 1-1. Then, you use a special editor known as the Blocks Editor to create the actions that the app performs. With the Blocks Editor, you do not have to know a language such as Java to program the app. Instead, you visually assemble *code blocks* to create the app's actions. Figure 1-2 shows an example of the Blocks Editor.

With App Inventor, you use a standard computer, like a Windows PC, a Mac, or a Linux system, to create an app. You can connect a supported Android smartphone or tablet to the computer either wirelessly or with a USB cable. As you develop the app, you will see it running on the connected device. (See Appendix B for more information about connecting your Android device to App Inventor.)

Figure 1-1 The App Inventor Designer (*Source:* MIT App Inventor 2, Pearson Education, Inc.)

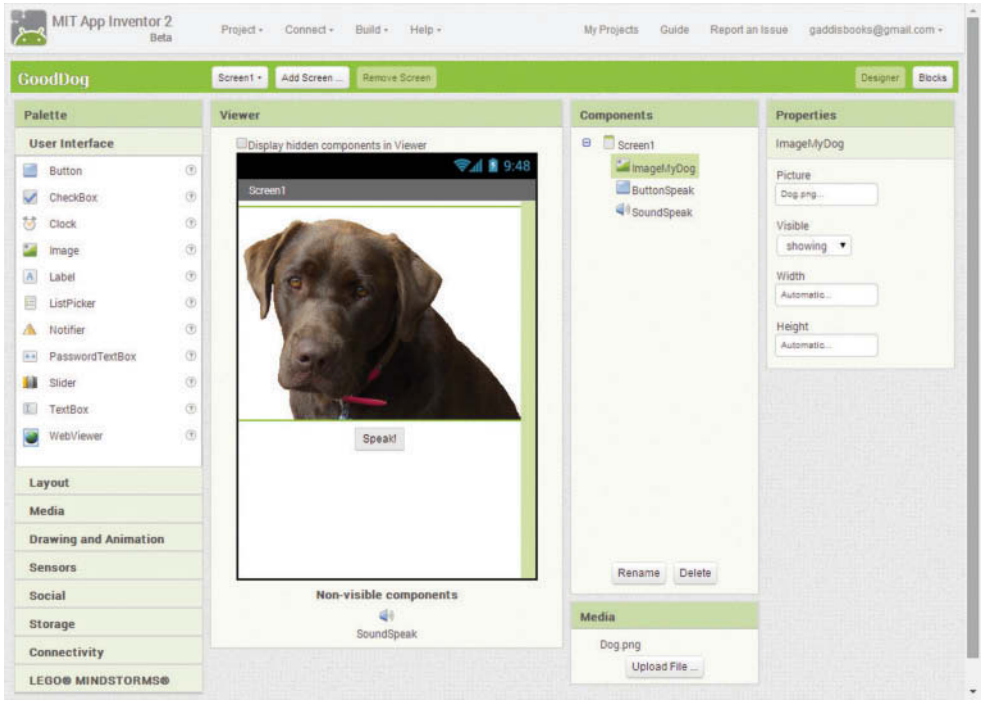
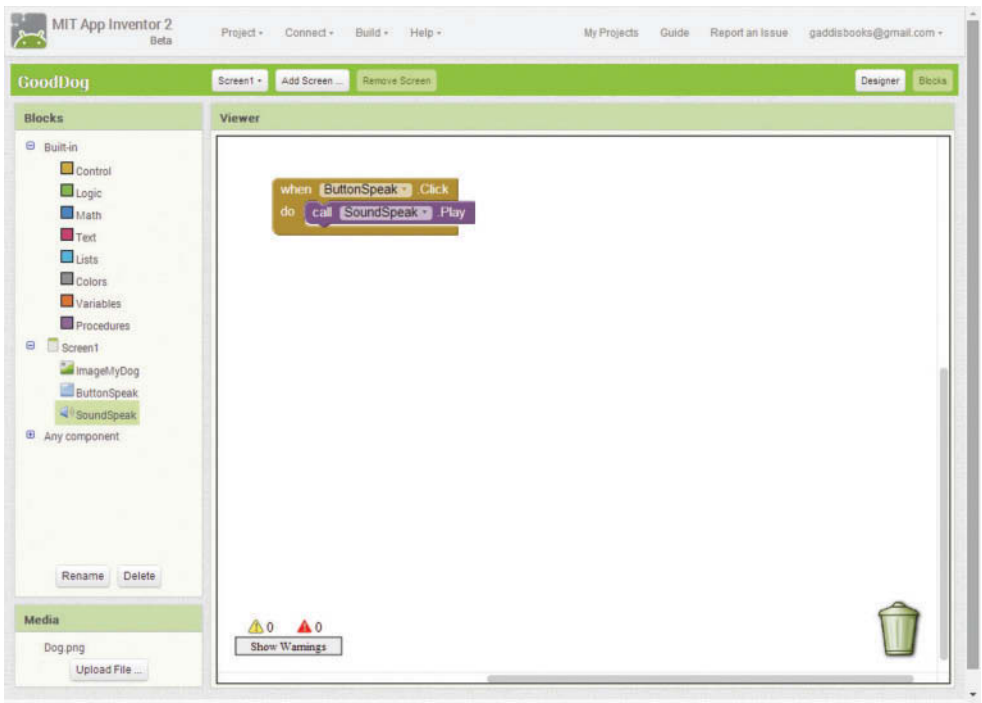
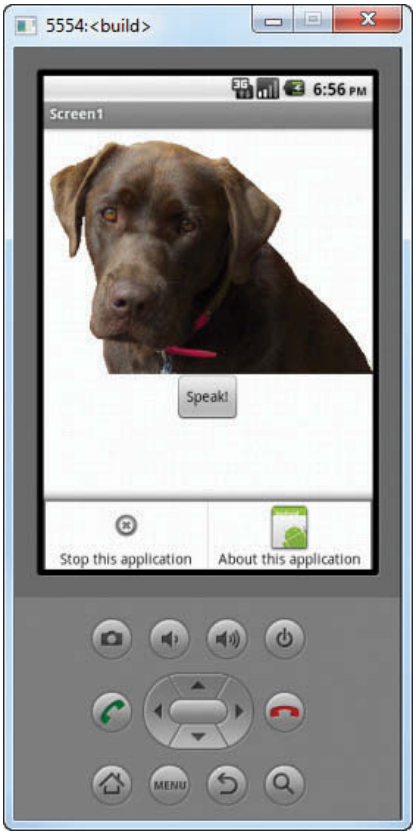


Figure 1-2 The Blocks Editor (*Source:* MIT App Inventor 2)



If you do not have a supported Android device to connect to your computer, App Inventor provides an Android emulator that runs on your computer. The emulator, which is shown in Figure 1-3, is a simulated Android phone. As you are using App Inventor to develop an app, the app appears and runs on the emulator's screen. You can interact with the emulator in many of the same ways that you interact with an actual smartphone. Although the emulator is limited (for example, it does not have a GPS sensor to report its location, and it cannot make phone calls), it does provide many of the basic features of an actual smartphone.

Figure 1-3 The Android Emulator (*Source:* MIT App Inventor 2, Pearson Education, Inc.)



App Inventor Runs in the Cloud

Although you will need to install a program on your computer to run the Android emulator, App Inventor runs in the *cloud*. This simply means that it runs on a remote server that you are accessing over the Internet. App Inventor is part of MIT's Center for Mobile Learning, so it is hosted on servers that are managed by MIT. Additionally, the projects that you create with App Inventor are stored on the remote servers.

There are several advantages to this cloud-based approach. For example, you can access App Inventor and your projects from any computer that is properly set up and connected to the Internet. In addition, the files that you create with App Inventor are maintained and backed up by the host. Also, you can be sure that you are always running the most recent version of App Inventor. Of course, this approach requires that you have an Internet connection to use App Inventor.

Setting Up App Inventor

Before you can work through the tutorials in this book, you must set up App Inventor to work with either the Android emulator or an actual Android device. If you haven't already done so, turn to Appendix A and follow the instructions to set up App Inventor. Appendix A also has an accompanying VideoNote that demonstrates the set up process. You can access the VideoNote from the book's companion website at `www.pearsonglobaleditions.com/Gaddis`. If you have an Android device that you want to connect to App Inventor, read Appendix B after you have set up App Inventor on your computer.

1.2 What Is a Computer Program?

CONCEPT: **A computer program is a set of instructions that a computer follows to perform a task.**

Before jumping straight into App Inventor, you should take a moment to learn some basic concepts about computer programming. The concepts that we discuss in this section apply to all types of computer programming, regardless of whether the computer is a laptop, a supercomputer, or a mobile device.

The title of this section poses the question "What is a computer program?" Before we can answer that, first we should answer the question "What is a computer?" To learn programming, you do not need a deep understanding of how computers work, but you do need to understand in the most basic terms what a computer is. Here's a definition that we can start with:

A computer is a device that follows instructions.

A computer doesn't know how to do anything on its own. It only follows the instructions that are given to it. Having said that, you must realize that a computer cannot follow just any kind of instruction. For example, you can't wake up in the morning and say to your computer, "Make an omelet and serve it to me in bed." That's not an instruction that a computer can understand. That's the kind of instruction that a human (like a butler, if you're lucky enough to have one) can understand. Unfortunately, common computers like the ones you and I have on our desktops don't make breakfast. Their purpose is to work with data. They do things

like adding and multiplying numbers, displaying data on the screen, storing data so it can be retrieved later, and so forth. Knowing this, we can expand our definition of what a computer is, as follows:

A computer is a device that follows instructions for manipulating and storing data.

When a computer is designed, it is equipped with a set of operations that it can perform on pieces of data. Most of the operations are very basic in nature. For example, the following are typical operations that a computer can do:

- Add two numbers
- Subtract one number from another number
- Multiply two numbers
- Divide one number by another number
- Move a number from one memory location to another
- Determine whether one number is equal to another number
- And so forth . . .

A computer instruction is merely a command for the computer to perform one of the operations that it knows how to do.

Although an instruction exists for each operation that a computer is able to perform, the individual instructions aren't very useful by themselves. Because the computer's operations are so basic in nature, a meaningful task can only be accomplished if the computer performs many operations. For example, if you want your computer to calculate the amount of interest that you will earn from your savings account this year, it will have to perform a large number of instructions, carried out in the proper sequence. Now we can understand what a computer program is:

A computer program is a set of instructions that the computer follows to perform a task.

So, if we want the computer to perform a meaningful task, such as calculating our savings account interest, we must have a *program*, which is a set of instructions. The instructions in a program must be carefully written so they follow a logical sequence. When a computer is performing the instructions in a program, we say that the computer is *running* or *executing* the program.

Algorithms and Programming Languages

Computer programmers do a very important job. Their job is important because without programs, computers would do nothing! When a programmer begins the process of writing a program, one of the first things he or she does is develop an algorithm. An *algorithm* is a set of well-defined, logical steps that must be taken in order to perform a task. For example, suppose we are writing a program to calculate an employee's gross pay. Here are the steps that should be taken:

1. Get the number of hours that the employee worked, and store it in memory.
2. Get the employee's hourly pay rate, and store it in memory.

3. Multiply the number of hours worked by the hourly pay rate and store the result in memory.
4. Display a message on the screen that shows the amount of money earned. The message must include the result of the calculation performed in Step 3.

Notice that the steps in this algorithm are sequentially ordered. Step 1 should be performed before Step 2, and so forth. It is important that these instructions are performed in their proper sequence.

The steps shown in the pay-calculating algorithm are written in English. Although you and I might easily understand the algorithm, it is not ready to be executed on a computer. The instructions have to be translated into *machine language,* which is the only language that computers understand. In machine language, each instruction is represented by a binary number. A *binary number* is a number that has only 1s and 0s. Here is an example of a binary number:

```
1011010000000101
```

When you or I look at this number, we see only a series of 1s and 0s. To the computer, however, this number is an instruction, which is a command to perform some operation. A computer program that is ready to be executed by the computer is a stream of binary numbers representing instructions.

As you can imagine, the process of translating an algorithm from English statements to machine language instructions is very tedious and difficult. To make the job of programming easier, special programming languages have been invented. *Programming languages* use words instead of numbers to represent instructions. A program can be written in a programming language, which is much easier for people to understand than machine language, and then be translated into machine language. Programmers use special software called *compilers* or *interpreters* to perform this translation.

Over the years, many programming languages have been created. If you are working toward a degree in computer science or a related field, you are likely to study languages such as Java, Python, C++ (pronounced "C plus plus"), and Visual Basic. These are only a few of the languages that are used by professional programmers to create software applications. Each of these languages has its own set of words that the programmer must learn in order to use the language. The words that make up a programming language are known as *keywords*. For example, the word `print` is a keyword in the Python 2 language. It prints a message on the screen. Here is an example of how the `print` keyword might be used to form an instruction in a Python 2 program:

```
print "Hello Earthling!"
```

This causes the message *Hello Earthling!* to be displayed on the computer screen. Compare this instruction to the binary number we saw earlier. You can see from this simple example why programmers prefer to use programming languages instead of machine language. Using words to write a program is much easier than using binary numbers.