LEARNING MADE EASY

Helping Kids with Coding

for dummies®

A Wiley Brand

Coach kids in coding even if you've never coded

Build young coders' skills in multiple coding languages

Help kids write code for apps, games, and gadgets

Camille McCue, PhD
Sarah Guthals, PhD

Helping Kids with Coding

by Camille McCue, PhD
Sarah Guthals, PhD

for dummies®
A Wiley Brand

Helping Kids with Coding For Dummies®

Published by: **John Wiley & Sons, Inc.,** 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Copyright © 2018 by John Wiley & Sons, Inc., Hoboken, New Jersey

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit https://hub.wiley.com/community/support/dummies.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at http://booksupport.wiley.com. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2018935055

ISBN 978-1-119-38067-2 (pbk); ISBN 978-1-119-38066-5 (ebk); ISBN 978-1-119-38058-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

Table of Contents

Introduction

Welcome to the world of computer programming! Whether you're an expert at programming or you've never written a line of code in your life, you can coach young people in learning the basics of coding. Just like learning to read, cook, or drive, basic principles define the discipline of coding, and the broader discipline from which coding is derived: computer science. This book coaches you step-by-step through the concepts and commands you need to help the kids in your life learn to program!

About This Book

Coding is fast becoming a skill that every child needs to be educated for in the 21st Century. Knowing how to code means possessing a skill that allows the children you're coaching to create things that are highly useful in modern society — apps, websites, analysis tools, and more. Helping kids learn how to code also means you're assisting them in developing a skill that is highly marketable and sets them apart from peers at school and later, in their careers.

But coding is taught at only a small fraction of schools, and often only at the high school level. This book offers you an easy-to-understand, but comprehensive, overview of all the coding fundamentals you need to teach. We largely avoid a theoretical approach to the material, instead offering you hands-on, practical content and methods of instructing your kids in coding. Like content in all *For Dummies* titles, this book is clear, concise, and organized in an easy-access format.

Helping Kids with Coding For Dummies is structured in a progressive sequence, with introductory topics preceding more challenging topics. The book builds in complexity, but you can dive straight in to any chapter, to discover more about that topic at any time. You already know about variables but need a bit of guidance in assisting your kid with loops? Then head straight to Chapter 10 for help.

As you explore each chapter, keep in mind the following structure:

- Each chapter begins with hands-on, away-from-computer activities. These games and "act-it-out" skits get you and your coders thinking about the big ideas before you dive into the code.
- Each chapter has guidance in writing code snippets in pseudocode, followed by many popular programming languages. *Pseudocode* is literally "fake code" that stands in conceptually for real code but doesn't really run on a computer.
- Each chapter ends with a small project that features the theme of the chapter. Projects are written in a variety of languages, but you can adapt them to any language in which you want to write them. You don't need to buy any software to use the featured programming languages.
- Programming code is shown in monofont.
- Command sequences using onscreen menus use the command arrow. For example, when working in Scratch, you can open a new project as follows: From the menu bar, choose File ➪ New.

Foolish Assumptions

In this book, we make some assumptions (possibly foolish assumptions!) regarding you getting started in your role as a coding coach:

- You are a parent, a teacher, an after-school guide, a summer mentor, a tutor, a coach, or other guide who is interested in helping youth learn to code.
- You have patience, a sense of humor, or both! Learning coding yourself — and helping young coders develop and improve their own skills — requires some of these.

We've also made some assumptions with regard to the coding work you'll be doing:

- You possess at least a little experience in typing on a computer keyboard, navigating a computer interface, and using a trackpad or mouse. Your background may be in using a Windows-based PC, a Macintosh, or both.
- You're capable of using a web browser such as Safari, Chrome, or Firefox, and you can type a URL to access a website such as Scratch.com and Code.org.
- You're comfortable with performing basic math, such as adding and subtracting, and performing basic logical operations, such as comparing two numbers.

» You can spell reasonably well, and you can locate and correct misspellings in your code. Programming languages often provide error messages as clues to help you track down misspellings, so you're not entirely on your own here! But you have to spell everything in your code correctly to get your program to run as you want.

Icons Used in This Book

The Tip icon indicates tips and shortcuts that you can use to make your work easier. These tips may apply to away-from-computer activities or to actual coding. Tips applying to issues that pop up often may be repeated in several places through the book.

The Remember icon calls your attention to important ideas you want to keep in mind while performing a task.

The Warning icon advises you to watch out, informing you of critical information that helps you steer clear.

The Technical Stuff icon marks slightly more in-depth nuts and bolts of programming, some of which might be helpful in achieving coding success.

Where to Go from Here

The programming concepts you use and coach are mostly universal. Because each concept features hands-on activities and pseudocode, you can teach these ideas to children as young as early elementary. But really, any age is the right age to start working on programming concepts, and there's never any wrong age to learn something new!

Progressing from away-from-computer concepts and pseudocode to at-the-computer coding can be done at any time when your young coder expresses interest. The programming snippets in this book are presented in some of the most popular languages used by novice coders, used from kindergarten through high school. All the languages are free, and all have stood the test of time with regard to their ease-of-use. Scratch is a popular way to get started, but for many

programming purists, Scratch's drag-and-drop puzzle pieces are inauthentic — their preference is any text-based language such as Python, JavaScript, or Java. Samples of all these languages are included in each chapter.

Don't forget to check out the cheat sheet that goes with this book. You can find what programming languages we recommend for each age group, all the projects we've created for this book, and more. Go to `www.dummies.com` and search by this book's title.

Regardless of which activities and programming languages you explore with your young coders, remember to cultivate curiosity, praise achievement, and encourage leveling up. And above all, have fun!

1 Getting Started with Coding

IN THIS PART . . .

Find out why kids are coding and how you fit into their journey.

Get the big picture of coding.

Find out which languages are the best for kids you're teaching.

Meet Steve Jobs, Steve Wozniack, Ada Lovelace, and Guido van Rossum.

IN THIS CHAPTER

- Why kids are coding
- Why you need to know coding
- Where do you come in?
- Working with young coders

Chapter 1

Welcome To (Or Back To) Coding

Who are you and where do you fit into the brave new world of coding? You may be a newbie programmer who wants to learn or "level up" coding skills to coach the next generation of kiddos to programming success. Or perhaps you're a seasoned programmer who wants to "dial it down" and explore a good starting point for kid coders. Or perhaps you're someone in between — you've coded in a past school or career experience — maybe in a language that's lost steam — and now you're returning to the practice to learn the newest tricks of the trade.

Whoever you are and whatever your goal, we're excited to welcome you to (or back to) coding!

In this chapter, you find out why kids are coding and why so much attention is currently focused in education on the discipline of computer programming. We also talk about the range of roles you can play in the teaching and learning of computer science, and identify strategies you can employ when working as a coding teacher, parent, or coach.

Why Kids Are Coding

Literacy has been a societal goal for centuries, with conscientious parents and teachers working to ensure that the children in their charge learn the skills necessary to succeed in their careers and in life. Until the 1970s, literacy meant mastering the traditional three "R's" of "reading, 'riting, and 'rithmetic" (spelling was considered less necessary). As technology started becoming commonplace, computers started appearing in educational settings, and tech literacy became viewed as the fourth literacy.

Fast-forwarding to the 21st century, technology has become so ubiquitous that not only is tech literacy a skill that makes you educated, it's a skill that makes you highly marketable in the workplace. While tech literacy can include general skills such as word processing, generating spreadsheets, and creating slideshow presentations, the real skills lie in computer programming, or coding. That's because coding allows people to be not just users of technology, but producers of it (at least on the software end of things).

Schools are recognizing that to prepare kids for their futures, a good education must include coding instruction. In some countries, including the United Kingdom and Canada, coding instruction is a national directive. In others, such as the United States, fewer than 10 percent of schools teach coding. To fill in the gaps, many online courses, after-school programs, and summer camps are providing kids instruction in coding. Just like learning to ski or learning a foreign language, learning the basics of coding is best accomplished at a young age: Every learning experience is "new and different" and it's easy to get back up when you fall down. Regardless of who is delivering the instruction, kids everywhere are coding — and you can help facilitate that learning with the kids in your life using the guidance provided in this book!

What are they learning?

Kids are learning more than just coding. They're devising solutions to problems, building games, and creating programs that do the routine and redundant work humans don't want to do. There's a lot kids have to learn to perform those tasks. Here's a quick rundown of what they're learning:

- **Computational thinking:** Computational thinking is the reasoning and planning you perform "in your head" when translating a problem and solution into a process that a computer can perform.

- **Algorithmic thinking:** Algorithmic thinking is mapping out an organized, efficient set of steps, which you can use and reuse to perform a task.
- **Communicating in a foreign language:** No, not an international language that's native to a country or people on Earth. But they are learning to communicate in a language that's not their native tongue — the language of computers. And they have to learn new words, new punctuation, and new rules for establishing successful communication.
- **Patience and resilience:** No matter what you're making — a computer program, a musical performance, or a gourmet meal — patience is required to learn a new skill, and resilience is required to bounce back from challenges to master that skill.
- **Creativity:** Contrary to popular belief, coding is not the cold, calculating discipline you may think it is. From inventing novel solutions to a problem, to inventing new video games, creativity is inherent in the coding process.
- **Troubleshooting:** Troubleshooting, or debugging code, means tracing and retracing your steps — sometimes by isolating and testing smaller sections of code, sometimes by tracing through the syntax, and sometimes by testing sample data to example the output — to find and fix problems.

How are they learning?

Kids are coding on their tablets, laptops, and desktops, using a variety of widely available software tools, many of which are free! They are using books, online resources, and tutorials in web and video formats, and discussions with friends to guide them.

Unlike the early days of coding, many of the programming languages and environments kids are using are visual in nature. Many offer tile or block-based formats in which kids can drag and assemble code blocks together like interlocking puzzle pieces to create programs. This type of structure allows kids to tinker without worrying about spelling commands correctly, syntax (grammar), or punctuation. Modern, introductory languages often feature built-in "assets" such as character costumes and sound effects. And they usually provide some sort of error reporting to help kids in their debugging.

If you tried coding before the 21st century, you probably learned text-based, also known as "line" coding; the new, visual ways of coding are most likely foreign to you. You may have used languages such as Basic, Pascal, COBOL, or FORTRAN. While these languages have mostly faded away in popularity, they were powerful

tools and popular in their heyday. Other "older" languages you may have used which are still around today include C, Python, Visual Basic, Ruby, Lua, and R. If you had the opportunity to tackle some serious coding, you may have worked with C++, Java, or JavaScript — some of the past and current heavy hitters in the coding world. The main kid-friendly language of yesteryear was Logo, what you may recall as the "turtle" language. Invented by MIT professor Seymour Papert, Logo and its derivatives were popular in schools, and still exist in several modern incarnations today. "Turtle" languages were about as close as most kids got to non-text based coding prior to Y2K. Fortunately, the more kid-friendly coding environments now available provide a lower floor for entering easily into the world of programming.

WARNING

Some purists are not fond of "kid" computer languages, especially those that are block-based, expressing concerns that this structure is not realistic, nor professional. They worry that these types of environments are not sufficiently authentic to lay proper foundations for future coding. However, research indicates that exploration in these languages still build the desired programming skills cultivated by working in more traditional, text-based, coding environments.

What does it mean down the road?

Your efforts in helping kids get started with coding lay the foundations for them to pursue more challenging programming activities in the future. You can help them build content skills, confidence, and the mindset required to succeed at coding. Whether they choose a career in computer science, or just dabble in writing small programs for various projects, your positive guidance and support contributes to developing an educated and confident young person. Who knows, you might even be responsible for cultivating the next Grace Hopper or Bill Gates!

COMPUTERS IN SCHOOLS

Kids have been coding since the first computers appeared in schools in the 1970s. Camille's school, Boone Elementary in San Antonio, Texas, received a teletype computer on which she had a 15-minute rotation once a week (although she often found ways to obtain additional time on the device). Operating essentially as a dumb terminal, the computer sent and received information over a simple telephone line, issuing drill-and-practice style math problems printed on giant rolls of form feed paper. Students would press the resistant, plastic buttons to input their answers. Following the completion of a problem set, the computer would respond with a score — at which time you would leap with joy if you earned 100%, or finagle more time to attempt another set if you had failed to reach the magical perfect score. Those were the days!

Why You Need to Know Coding

You have this book in hand, but you may still be asking yourself why you need to know coding. Why should you learn to code when your young coder has a teacher or a camp instructor or a YouTube video guide?

You should learn to code for many reasons:

- **The more you know, the better you can help.** More content knowledge and more relevant experience on your end leads to better ability to coach your young coder. This is especially true when it comes to troubleshooting code: kids make the same mistakes you make; if you've "been there, done that," you can help them find and fix their errors.
- **The more you code, the more you can empathize with your coder.** Sharing the joy and frustration of coding is part of being a good coach. The more you experience these emotions yourself, the better you are at understanding and appreciation the mindset and affective disposition of your kid.
- **It's the fourth literacy.** Maybe coding was offered when you were in school. Maybe it wasn't. Either way, it's here now and it's here to stay — and as the "fourth literacy," you need to learn it to be fully educated and a full participant in today's world.
- **It's quality time with your kids.** If you're a parent, you do a lot with your kids, but sometimes that "doing" is spent more passively than you'd like (driving to soccer practice, buying school supplies at Walmart). Learning something new (coding!) and working together on an app or website provides a great opportunity for you to spend quality time with your kids.
- **Doing so allows you to practice what you preach.** You're a teacher, a parent, a coach. . . and you're also a role model. If you don't already know how to code, it may be a challenge to encourage someone else to do so. As they say at Nike, Just Do It!

Fear and loathing (of coding)

One of your biggest challenges in learning to code may be your own fear and self-doubt. Perhaps you think you're "not smart enough." Another challenge may be a genuine dislike or disinterest in code. Like any new endeavor, there is often a sense of concern that you won't be capable of learning something new. Maybe you won't like it. Maybe it's too much work.

It may be worthwhile to note that a lot of other people who came before you shared the same fears (and loathing, of course) of coding. Like learning any new field — playing piano, ice-skating, speaking Mandarin, cooking, gardening, sewing — there is a learning curve in which the introductory phases are not especially fun or rewarding. But hopefully the experiences leading up to adulthood have shown you that, over time, sticking to the process of learning a new skill eventually results in elevated abilities and satisfaction in a job well done.

Computer programmers are not smarter than you are; they've just been at it longer! Like you, they started with introductory coding, building their skills a bit at a time, learning new programming languages and writing many programs until they built a solid base of coding knowledge and skills. Congratulations on taking the first step of learning to code and coaching the next generation to early successes in the world of computer science.

You may already know more than you think

If you're panicking that you suddenly need a degree in computer science to learn coding to successfully help the kids in your life. . . don't! You may already know more about coding and its underlying principles than you think. Just ask yourself a few questions:

- Are you good at learning new languages? Because coding uses its own languages featuring vocabulary, grammar, and syntax — you may find that learning to code is a snap!
- Are you good at planning and organizing? Because coding is about using concise, reusable instructions, start-to-finish — you may find that learning to code is familiar and easy!
- Are you confident about tinkering with your computer or mobile device to learn and use new features? Because coding is about commanding your technology devices to bend to your will — you may find that learning to code is easy and empowering!
- Are you excited by new challenges? Because coding energizes you to invent something tangible from nothing more than an idea in your head. You may find that learning to code is an exciting new adventure!

Sarah has taught hundreds of adults to code for the purpose of engaging the kids in their lives in coding. You can do it, too!

Where Do You Come In?

You come in by guiding and supporting young people in their coding endeavors, as a classroom teacher, a camp or after-school coach, or a parent/mentor. You do not have to be an expert coder — just an interested and caring adult who is willing to co-learn and support your kids in their pursuit of the computer science mission! Here are some ways you can accomplish these feats.

In the classroom

You can select a variety of coding experiences for a classroom setting depending on factors including grade level, available technology, and expected contact time. In the classroom setting — which is more formal than other settings — where you may meet with students multiple times over a quarter, semester, or full school year, you likely have time to work on developing both a breadth (covering many topics) and depth of programming skills among your students (providing students time to grow a greater complexity of skills within a topic of focus). You want to choose a programming environment, a curriculum, and appropriate technology tools for getting kids coding in your classroom.

Programming environments

You can choose several excellent programming environments for coding instruction. Most are free, but be sure to check online for the latest information and updates on each product.

- **Everyone, away from the computer:** Not all computer programming work needs to occur on a computer. Both Computer Science Unplugged (csunplugged.org) and Code.org provide extensive, hands-on, away-from-the-computer activities to teach computational thinking concepts. These activities are superb lead-ins to their technology analogues, and are likely to be hits in their own right!
- **PreK:** Daisy the Dinosaur and Bee-Bots are perfect for three and four year olds. Both feature a very limited set of commands, along with cute characters for the pre-reading crowd. Bee-Bots also has the advantage of providing a physical device for students to program and navigate around a playmat that you spread on the ground. The coding paradigm of both is drag-and-drop tiles. Daisy is available in the App Store for use on iPhone and iPad devices. Bee-Bots can be used on Mac and Windows platforms, as well as iPhones and iPads.

- **K-Grade 1:** Scratch Jr. and the Foos are both coding environments that visually pop and are extremely rich in pre-reader coding experiences. The Foos presents a series of puzzles that increase in complexity, while Scratch Jr. offers a more open-ended exploration space. The coding paradigm of both is drag-and-drop tiles. Both the Foos and Scratch Jr. run on mobile devices (iOS and Android) and in web browsers.
- **Grades 2–5:** Scratch and Tynker offer deep, exciting, coding experiences through which students can explore critical coding concepts, express their creativity, and share their work publicly. The coding paradigm of both is drag-and-drop tiles. Both run online and can also be downloaded and used in an offline environment.
- **Grade 6 and up:** Python, JavaScript, and GameSalad environments are all quite different, but they're a step up in complexity and offer real-world programming experiences to your young coder. Python is used in a variety of contexts, including commanding robots. JavaScript provides interactivity on websites, occupying a special place in web pages formatted using HTML and CSS. This book also addresses the use of Python and JavaScript in coding for the micro:bit electronics board — a great tool for applying your code to Internet of Things (IoT) devices. GameSalad facilitates creation of authentic game apps that can be sold in the App Store and Google Play store (but you have to pay to obtain a GameSalad license).
- **Grade 9 and up:** Java, C++, and Swift are considered professional coding languages that your coder can migrate to as she elevates her programming prowess. They are used for authentic programs and for writing software for IoT devices. Java is currently the language used on the AP Computer Science A exam — your coder is probably going to see this before she leaves high school. C++ is used in a variety of applications, such as databases and video gaming. Swift is Apple's powerful and easy-to-use programming platform.

  App Inventor is a drag-and-drop teaching language for mobile devices. It provides a higher degree of complexity than other "easy to use" environments, while at the same time offering the use of Application Programming Interfaces (API), which are pre-written software that allows two applications to talk to each other. App Inventor permits your coder to do cool, "real" things such as integrate Google Maps and GPS location in apps.

Turn to Chapter 3 for more detailed information about each of these programming environnments and others that may be of interest to your young coders.

Choosing curriculum

The curriculum you select for your classroom depends on a variety of factors. You may be required to follow school or district guidelines created by educators other

than yourself. Or perhaps you must adhere to standards set at the state level, or the national level by a group such as CSTA (Computer Science Teachers Association) or ISTE (International Society for Technology in Education).

Remember, coding is about creating authentic products that perform real tasks. Crafting your curriculum in a project-based model helps ensure students are doing the type of work that professional computer programmers perform. Try to create and customize activities in which students make real products, producing products including websites, online games, and apps. Encourage and support their innovations in crafting inventive graphics, multiple levels, and other customizations that make each child's program stand apart from her peers.

Most curriculum is shaped by a scope and sequence that ensures learners grow in their coding skills to meet key goals, with short-term benchmarks established along the way. Classroom teachers still have the opportunity, and the responsibility, to differentiate instruction for students of varying ability levels to appropriately challenge each student to rise above his current level. Create tiered options in which each student can create programs that match his skill set. Your coding experience and understanding of each individual child can help you customize the coding experience to create a successful learning environment for your entire classroom community. For specific coding curricula and associated grade level designations, check out Part 5.


TIP

Training and professional development (PD) for specific curricula are offered by organizations and universities. For example, the University of Texas at Austin (Camille's alma mater!) provides online and in-person PD for its UTeach AP Computer Science Principles course. The College of St. Scholastica offers an all online, four-course certification in CS Education. Such PD can result in credit to educators for things like raises or certifications.

Camp or after-school coach

With the increased attention on coding, more and more after-school programs and summer camps are popping up everywhere. If you're an instructor in such a program, you have a unique role in helping the young coders in your charge. You probably don't see the kids as consistently as a classroom teacher. And there may be greater variation in the experience and ability level of your kid community. But it's also probable that you're not locked into a highly structured curriculum.

Whatever the content and format of your workplace, remember to teach students first and coding second. Take time to learn each coder's interests, experience, and programming goals. Ideally, take a few notes on each participant, update them following each coding session, and review them prior to your next meeting.

Reminding yourself regularly of how each coder is progressing can help you to assist him in moving forward and reaching his goals.

As frequently as possible, communicate progress to your coder's parent or guardian. Include samples of the work product, especially links to completed programs the coder's family can view online!

Mentor

Mentors foster and grow coding abilities in youth through a variety of informal contexts. Perhaps you're a parent, friend, or co-worker of someone with a child who wants to learn coding. Or perhaps you're the teacher whom "everyone comes to see" when students need assistance on special projects they're tackling. Maybe you're a professional programmer who volunteers your time at Girl Scouts or Boys Scouts workshops. (Both groups have coding badges that kids can earn!) Mentors often work on an as-needed basis helping kids grow their coding skills or track down an extra-hard-to-find-bug. They typically maintain a long-term partnership with the kids they mentor, suggesting new projects, languages, and courses to pursue. As a mentor, you may also be asked to write letters of recommendation about the coders in your charge, attesting to their coding interests and abilities, to help them gain admittance to special programs in high school or university.

STEVE WOZNIAK AND STEVE JOBS: STARTING THE PERSONAL COMPUTER REVOLUTION

Commonly known as the Apple Guys, Steve Wozniak and Steve Jobs started the personal computer revolution, working on a great idea out of a garage. The products they designed and brought to life made home computing possible, easy, and relatively inexpensive for the first time in human history.

Both men were born in the 1950s and grew up during the decade when several electronics and computing firsts occurred: The first integrated circuit and the first computer modem were invented, the programming language FORTRAN was developed, and the first transistor radio was produced by Texas Instruments. These advancements set the stage for the work Wozniak and Jobs initiated in the early 1970s.

Introduced by a common friend who knew they liked electronics and loved pranks, Wozniak and Jobs initiated a working relationship with their creation of a digital tone

generator. Known as the Blue Box — a rather illegal device — when connected to a telephone, it allowed users to make free phone calls, even internationally! More importantly, Jobs and Wozniak found that they enjoyed collaborating on technology projects, and both had a vision that computers weren't just for big businesses. They believed that an affordable, desktop-size computer could exist in every home, and that it could be used for everything from creating artwork, to managing budget spreadsheets, to playing games. The two men began designing and building their initial home computers, with Jobs focusing on the business and marketing side of the venture, and Wozniak (or "Woz" as he is called) focusing on the engineering design and technology.

Once Jobs and Woz realized there was demand for the first personal computers they built and sold, they landed an investor and the Apple Computer Company was launched! Known for their exceptional design qualities including an easy-to-use graphical user interface (GUI) and simple, elegant casings, Apple computing devices quickly established a permanent foothold in the world of home computers. As new versions of Apple's computing products emerged, the company grew and spawned new products — iPods, iPads, and iPhones — as well as exciting Mac stores where people, young and old, can purchase Apple products and learn how to use them.

When Jobs and Woz started the computer revolution, they envisioned having a computer on every desktop. If you have a Mac desktop, a MacBook laptop, or an Apple mobile device, then you've helped make their dream a reality! As Steve Jobs once said, *"The people who are crazy enough to think they can change the world are the ones who do."*

Photo credit: https://alumni.berkeley.edu/california-magazine/spring-2015-dropouts-and-drop-ins/silicon-valley-s-merry-prankster-put-his

Working with Young Coders

There are wide range of dispositions when it comes to kids getting started with coding. . .here are a few of the personality types you may encounter:

- **The skydiver type:** This type of kid wants to dive right in, trying every line of code, without any specific plan of action associated with his efforts. While we applaud confidence and creativity at the computer, we would encourage you to steer this kid towards thinking and planning prior to coding and executing. A few years ago, a new child at Camille's school was very proud of his Scratch "expertise" — which turned out to be dragging hundreds of Scratch tiles and assembling them in nonsensical ways in the program workspace. It took several months to help him learn how to evolve from chaotic habits to conceptualizing an idea and translating it into functional code. Helping this type of child to work procedurally and incrementally, taking pride in completing a project start-to-finish, are behaviors you want to foster.
- **The rational actor:** This kiddo wants to learn a new concept and then try out the associated code at the computer, one step at a time. This type of learner usually experiences success, but may struggle when confronted with an information gap in which she needs a command that she hasn't previously encountered. Helping this child to branch out and research on her own, employing a bit of grit in finding a solution, is a habit you want to coach and develop.
- **The happy passenger:** This child is a bit tentative to try coding — even if he is a game-player or social media guru! He may seem in command of tech until he has to pop the hood and get into the mechanics of writing actual code. He has some tech-savvy, but until this point, he was just happy to be along for the ride. You need to help him understand that you're going to support his efforts to transitioning from a tech user, to a tech maker — through learning to code.
- **The next Bill Gates:** This kid loves coding, knows everything about coding, wants to do more coding, watches Silicon Valley, has built her own computer, and has already applied to the Stanford Computer Science undergrad program. Continue to cheer for this kid, help clear the runway for her by removing trivial obstacles (which often exist in school settings), and actively seek projects, competitions, and peer programmers (who may be older) as partners for her.

Regardless of which kids and which coder personalities you encounter (likely all of them!), meet them where they are, and help lift them up to the next level. You have an important role in fostering their coding foundations and building both the hard and soft skills of a coder. Onwards!

IN THIS CHAPTER

- Coaching for coding
- Thinking algorithmically
- Commenting to plan and document programs
- Understanding sequencing, repetition, and selection

Chapter 2
Understanding the Big Ideas

This chapter helps you better understand the big concepts in coding, or computer programming: giving instructions to a computer so that it can perform a task. You see the big picture of writing a program before you focus on the specific details of writing lines of code.

Seeing the Big Picture in Coding

Learning to program a computer is similar in many ways to playing football. You have to first think about the goal of the program. Then you focus on the big picture, or game plan, to develop the key parts of the program.

For example, the goal of football is outscoring your opponent. The big picture of football may include kicking the football, running with the football, avoiding getting tackled, and moving the football to the end zone. Drilling down into special plays and perfecting fancy footwork comes after players cement their understanding of the goal and the big picture.

In coding, the goal is to complete a process that would not easily be accomplished without a computer. This may be providing an airplane simulation game or a tool for searching for a home overseas. The big picture involves all the large parts of the program that contribute to achieving the goal. Creating an airplane simulation may consist of providing a user a virtual airplane, giving controls to fly the plane, providing environments for the plane to fly in, and making the plane react according to user input. Creating a tool for searching for a home overseas may consist of providing the user a map, building areas onscreen to enter information about the desired attributes of the home, and creating a search mechanism based on user-input attributes.

How can you represent the big picture? You can act it out, or draw pictures, or write words to describe the important parts. Each of these activities is *unplugged* — they don't require the use of a computer — and each one helps you plan out your computer program before you sit down to begin coding.


TIP

Ask your young coders to think of something they're good at, such as baking cookies, playing Minecraft, or taking care of the family pet. Then ask them to think about their special skill and how they would explain it, in very simple terms, to an alien who has just stopped in to visit Earth. What are the most important parts of understanding how to go from ingredients to yummy baked cookies? What are the big picture ideas you need to know to complete a successful dog walk?

Acting Out the Big Picture, Unplugged

Too many times, people dive into a project by focusing on the details before mapping out the big picture. Camille once had a scary assignment in fifth grade in which she made this mistake! When asked to write a summary of *The Red Badge of Courage*, she panicked and tried to write a detail from every page of the book, somehow stitching them together, but not making any sense at all. It was an awful three hours of tears at the dinner table. What she should have done was take a step back and write a 30-second movie trailer version of the book, hitting the highlights and the most important parts that captured only the main ideas.


WARNING

Thinking about coding in terms of typing commands on a computer without stepping back to consider the big picture of what you want the program to do is a recipe for disaster. Plan before you code!

Dramatizing a noncoding process

One way of understanding the big picture of a computer program is to act it out, without any computer at all. Invite your young coder to try acting out an everyday activity that is not computer related.

One possibility is dramatizing the process of washing socks (as if asking a kid to wash her own socks wouldn't be drama enough!). For example:

1. **Ask your young coder to take off her socks (or dig them out of the laundry basket!).**
2. **Put the socks in the washing machine and pretend to wash them.**
3. **Take them out of the wash and move them to the dryer where you pretend to dry them.**
4. **Remove the socks from the dryer (all clean and dry!) and ask her to put them on her feet.**

Although you call this process "doing laundry" or "washing socks," you can have her act it out so that she can see the process actually involves more steps (see Figure 2-1).

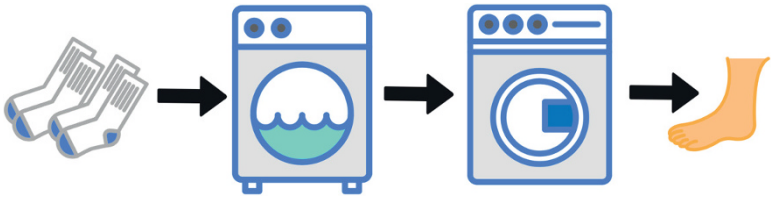

FIGURE 2-1: Acting out a process, such as washing socks, helps kids better understand steps in a process.

"Drive to the intersection and turn right" is an example of a similar task you may perform when writing code for a car race video game. Although this single phrase describes the big picture, you need to drill down to more specific steps in order to accomplish the bigger task. For example, you need to move forwards 150 pixels and then make a square turn (also called a 90 degree turn) to the right (or to the left). When coding these actions in a programming language, you can use a sequence of commands that looks something like `forward 150 right turn 90`.

Acting out a big picture doesn't have to involve props, but it's often helpful to use them with new programmers. Making concrete connections to everyday objects and processes can help your kid form a mental model that she can refer to when she writes a computer program.

Walking through some daily tasks

TIP

You can step through a number of daily tasks to help a young learner grasp the concept of process. Brainstorm a list of everyday processes with your child and then ask him to demonstrate the big picture of each process. Here are some you can try:

- **Get up in the morning.**
  1. Alarm clock rings.
  2. Wake up.
  3. Swing legs over the side of bed.
  4. Stand up.
- **Feed the dog.**
  1. Call the dog.
  2. Scoop the dog food from the bag.
  3. Place food in the bowl.
- **Perform a cannonball.**
  1. Stand at the edge of the pool with arms outstretched.
  2. Leap up and out over the water while tucking legs into your chest.
  3. Hit the water, tushie first.
  4. Cheer with joy as displaced water booms into the air, splattering everyone in a 3-meter radius.
- **Make a smoothie.**
  1. Gather the fruit.
  2. Put fruit into the blender.
  3. Add ice.
  4. Mix.
  5. Pour into a glass.

Get creative and see how many processes you can act out! Remember, you don't have to actually use the real materials to act out a process. As in charades, you can use your imagination!

Creating an Algorithm

Think about the steps you use to perform any type of process — for example, getting up in the morning, performing a layup in basketball, or washing socks in the laundry. You probably perform the same actions every time. That's because a specific set of steps, from start to finish, define that process. Anyone who performs the same process probably uses those same steps, or a series of steps, that are nearly identical. Those steps are called an *algorithm.*

An *algorithm* is a step-by-step set of instructions to be followed in completing a task, especially by a computer.

Computers are fast, and they don't get bored or annoyed with doing complicated math or performing the same tasks over and over again. But they aren't especially smart, and they don't think for themselves (yet!), so it's up to the human operator of the computer to give it an algorithm so that it knows how to perform a process.

Turning a picture into words

One algorithm kids have likely performed in the household (or that adult family members wish they would perform!) applies to vacuuming. Unless you are living with a random vacuumer — someone who uses no specific pattern when attacking the carpets with the household Hoover — then you probably see a pattern used to vacuum rooms. What does the vacuuming algorithm look like? As a picture, it probably looks something like Figure 2-2.

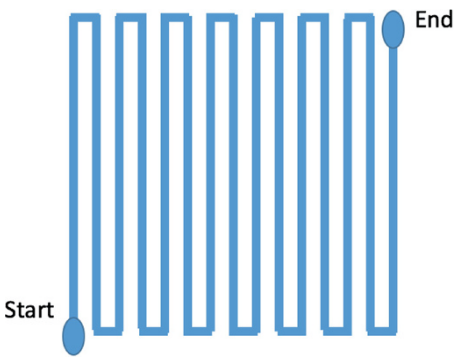

FIGURE 2-2: A visual representation of a common vacuuming algorithm.

Turning the picture into words requires thinking about each step, from start-to finish, that the designated vacuumer needs to do to clean the room. These words, in order, may look like the following steps:

1. **Position the vacuum in one corner of the room.**
2. **Plug in the vacuum.**
3. **Turn on the vacuum.**
4. **Push the vacuum in a straight line.**
5. **If you hit a wall, make a corner turn towards unvacuumed area.**
6. **Push forward a little.**
7. **Make a corner turn towards unvacuumed area.**
8. **Repeat Steps 4 through 7 until you reach the wall opposite your starting point.**
9. **If you've reached the wall opposite your starting point, then turn off the vacuum.**
10. **Unplug vacuum.**

The step-by-step vacuuming process is a common algorithm for cleaning up your carpet, but the zigzag pattern isn't the only possible one. What other vacuuming patterns can you think of? What about patterns that start with the vacuum in the center of the room, such as a spiral pattern, or a pattern that looks like the spokes of a wheel?

What would the algorithm look like for each of these vacuuming patterns in Figure 2-3? Try to write them!

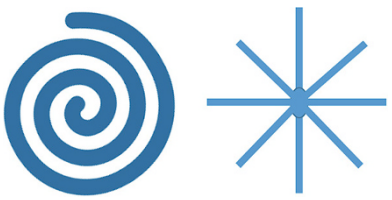

FIGURE 2-3: A spiral or a starburst pattern can be the basis for alternative vacuuming algorithms.

One possible vacuuming algorithm in code

Your young coder may already be asking, "So how can I turn this picture algorithm into code?" Here is a simple program in Scratch that shows one possible vacuuming algorithm. (For more on Scratch, see Chapter 3.)

The Scratch *stage* shows the room where the vacuuming takes place. On the stage are three objects: a front wall, a back wall, and a vacuum. The vaccum is initially positioned in the lower-left corner of the stage. Vaccuming progresses from the left wall (where the x-coordinate is large and negative) to the right wall (where the x-coordinate is large and positive). The front and back walls are used as indicators to know which direction to turn the vacuum. The vacuuming algorithm executes in a forever loop: If the vacuum touches the front wall, it turns right; if the vacuum touches the back wall, it turns left. The following command is used to find out whether the vacuum has reached the wall opposite the starting point:

```
if x position of vacuum > 220
```

When it does, the vacuuming process is complete, and the program ends by executing a `stop all` command, as shown in Figure 2-4.

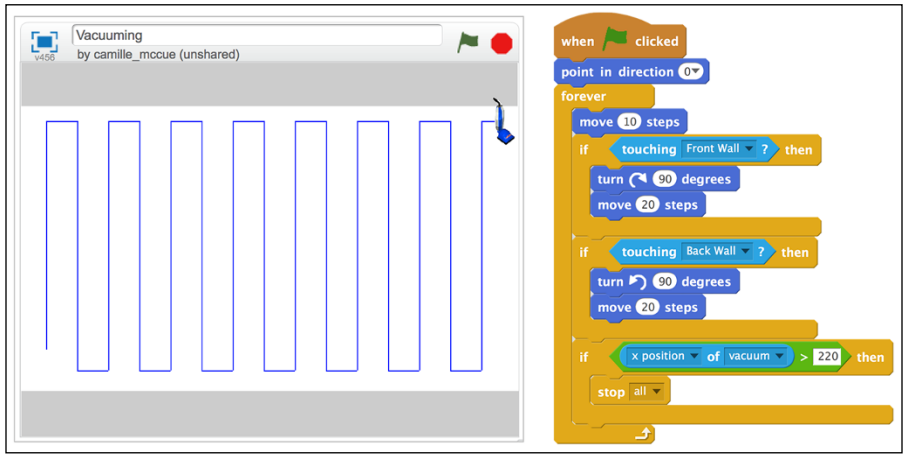

FIGURE 2-4: The vacuuming algorithm written in Scratch.

Don't worry about the details of how to write the code; just see whether you can trace the algorithm. Notice at the start of the program that the vacuuming event begins when the green flag is clicked and that the starting conditions of the vacuum are also set at the start of the program: The vacuum points in the direction of 0 degrees — the front of the room — and (outside of the program) the user manually positions the starting point of the vacuum.

In later chapters, you can work more on translating algorithms into code. This chapter focuses on helping your coder assemble different types of commands together into a larger program that performs a task.