

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

Introduction	1
Part 1: Getting Started with Coding	5
CHAPTER 1: Welcome To (Or Back To) Coding	7
CHAPTER 2: Understanding the Big Ideas	19
CHAPTER 3: Figuring Out Programming Languages	41
Part 2: Getting Your Hands on Code	61
CHAPTER 4: Working with Words	63
CHAPTER 5: Knowing Where You Are. . . and Where You're Going	81
CHAPTER 6: Getting Fancy with Graphics and Sound	107
Part 3: There IS Math on This Test!	125
CHAPTER 7: Tackling These Ever-Changing Variables	127
CHAPTER 8: Computing Using Math	145
CHAPTER 9: Helping with Logic Operations	167
CHAPTER 10: Getting Loopy	185
CHAPTER 11: Adding Lists	201
CHAPTER 12: Coding Subprograms	221
Part 4: Applying What You Know	237
CHAPTER 13: Fixing Problems by Debugging	239
CHAPTER 14: Creating a Webpage	255
CHAPTER 15: Building a Mobile Game	289
CHAPTER 16: Programming Simple Electronics	317
Part 5: The Part of Tens	337
CHAPTER 17: Ten Do's and Don'ts for Selecting a Kids Coding Curriculum	339
CHAPTER 18: Ten Ways to Keep the Coding Learning Going	349
Index	357

Table of Contents

INTRODUCTION	1
About This Book	1
Foolish Assumptions	2
Icons Used in This Book	3
Where to Go from Here	3
PART 1: GETTING STARTED WITH CODING	5
CHAPTER 1: Welcome To (Or Back To) Coding	7
Why Kids Are Coding	8
What are they learning?	8
How are they learning?	9
What does it mean down the road?	10
Why You Need to Know Coding	11
Fear and loathing (of coding)	11
You may already know more than you think	12
Where Do You Come In?	13
In the classroom	13
Camp or after-school coach	15
Mentor	16
Working with Young Coders	18
CHAPTER 2: Understanding the Big Ideas	19
Seeing the Big Picture in Coding	19
Acting Out the Big Picture, Unplugged	20
Dramatizing a noncoding process	21
Walking through some daily tasks	22
Creating an Algorithm	23
Turning a picture into words	23
One possible vacuuming algorithm in code	24
Representing Algorithms	26
Acting it out	27
Drawing a picture	27
Creating a storyboard	28
Building a flowchart	28
Writing pseudocode	30
Commenting the bones	31
Organizing with Sequence, Selection, and Repetition	33
Sequence	34
Selection	35
Repetition	36
Including Randomness in Your Coding	38

CHAPTER 3: Figuring Out Programming Languages	41
What You Want in a Language	42
Free Languages for Tots and Kids	42
The Foos	42
Think & Learn Code-a-Pillar	43
Daisy the Dinosaur	43
Scratch Jr.	44
Free Languages for Youth and Tweens	45
Scratch	45
Hopscotch	47
Kodu	47
Languages for Teens and Older	48
Alice	48
MIT App Inventor 2	49
Python	50
JavaScript	53
Java	55
Other Awesome (Not-So-Free) Languages	58
MicroWorlds EX	58
Tynker	58
GameSalad	58
 PART 2: GETTING YOUR HANDS ON CODE	 61
CHAPTER 4: Working with Words	63
Communicating with Text	63
Showing Text Onscreen	64
Using pseudocode	64
Using Scratch	64
Using Python	65
Using HTML	66
Using JavaScript in an app	66
Using Java	68
Words In, Words Out	69
Using Scratch	70
Using Python	71
Using HTML and JavaScript	71
Using JavaScript in an app	72
Combining Text Onscreen	74
Using pseudocode	75
Using Scratch	75
Using Python and other languages	75
Formatting Text Onscreen	77
A Mad Libs Example	78

CHAPTER 5:	Knowing Where You Are . . . and Where You're Going	81
	Acting Out Position, Unplugged	82
	Setting and Finding Position	85
	Using pseudocode	85
	Using Scratch to set position	86
	Using Scratch to find position	87
	Using JavaScript	87
	Positioning Objects Randomly	93
	Using Scratch	93
	Using JavaScript	94
	Setting and Finding Direction	95
	Using pseudocode	95
	Using Scratch	96
	Setting Object Direction Randomly	97
	Using Scratch	97
	Turning	98
	Using pseudocode	98
	Using Scratch	98
	Acting Out Motion, Unplugged	99
	Making an Object Move	100
	Using pseudocode	100
	Using Scratch	101
	Using JavaScript	103
	Asteroid Blaster	104
CHAPTER 6:	Getting Fancy with Graphics and Sound	107
	Sizes of Images and Sounds, Unplugged	108
	Activities surrounding images and sounds	108
	Knowing your sizes	109
	Using Graphics in Your Programs	109
	Image file types	109
	Creating images	110
	Finding images on the web	111
	Importing a JPEG or PNG in Scratch	114
	Importing a GIF in Scratch	116
	Importing a JPEG, PNG, or GIF in JavaScript	117
	Adding Sound to Your Programs	117
	Sound file types	118
	Creating original sounds	118
	Finding sounds on the web	119
	Importing sounds into Scratch	120
	Importing audio into JavaScript	121
	Creating a Sound Board	122

PART 3: THERE IS MATH ON THIS TEST!	125
CHAPTER 7: Tackling These Ever-Changing Variables	127
Acting Out Variables, Unplugged	127
Variable parts	128
Dramatizing variables	130
I Do Declare (And Initialize)	132
Using pseudocode	132
Using Scratch	133
Using Python	134
Using JavaScript	135
Using Java	136
Checking on Variable Values	137
Using Scratch	138
Using Python	138
Using JavaScript	138
Using Java	140
Incrementing and Decrementing Variables	140
Using pseudocode	140
Using Scratch	141
Using Python	141
Using JavaScript	142
Using Java	142
Creating a Stock Ticker	142
CHAPTER 8: Computing Using Math	145
Acting Out Math, Unplugged	145
Number types	146
Dramatizing math	146
Doing Simple Math	149
Using pseudocode	149
Using Scratch	149
Using Python	150
Doing Advanced Math Operations	150
Using pseudocode	151
Using Scratch	152
Using Python	153
Oh So Mod — Using the Mod Operation	156
Using pseudocode	157
Using Scratch	157
Using Python	157
Ordering Those Operations (PEMDAS)	157
Using Scratch	158
Using Python	158

Rounding	159
Rounding via casting in Java.	160
Rounding decimals to integers via methods.	160
Generating and Using Random Numbers	162
Using pseudocode.	162
Using Scratch	162
Using Python	163
Coding a Crypto Code Maker.	163
CHAPTER 9: Helping with Logic Operations	167
Simple Logic, Unplugged	167
Programming Simple Conditionals.	169
In pseudocode	169
In Scratch	169
In Python.	170
In JavaScript	170
In Java	172
Advanced Logic, Unplugged.	174
Coding Compound Conditionals (aka, AND, NOT, and OR Will Get You Pretty Far!)	176
In pseudocode	177
Compound conditionals in Scratch.	177
In Python.	179
In JavaScript	181
In Java	181
Rock, Paper, Scissors.	182
CHAPTER 10: Getting Loopy	185
Loops, Unplugged	185
Repeat fun, unplugged	186
Random loop conditions, unplugged	186
Loop Types and Structures	187
Infinite loops.	188
Actions repeated in loops.	188
Conditions of loops	188
Using pseudocode.	189
Using Scratch	191
Using Python	193
Nesting Loops	196
Using pseudocode	196
Using Scratch	197
Using Python	198
Coding the Classic Fibonacci Sequence	199

CHAPTER 11: Adding Lists	201
Lists, Unplugged	201
Introducing Lists	203
Using pseudocode	203
Using Scratch	205
Using Java	208
Sorting Lists	215
Selection sort: An easy sorting algorithm	215
Common application: Arranging numbers in order	216
Searching Lists	217
Linear versus binary searching algorithms	217
Common application: Finding a phone number	218
CHAPTER 12: Coding Subprograms	221
Subprograms, Unplugged	221
Starting with Pseudocode	223
Creating a Spirograph with Subprograms	224
Pseudocode	225
Scratch	225
JavaScript	227
Java	228
Coding Subprograms with Parameters	230
Scratch code block with parameters	230
JavaScript, with parameters	233
Java, with parameters	234
PART 4: APPLYING WHAT YOU KNOW	237
CHAPTER 13: Fixing Problems by Debugging	239
Debugging, Unplugged	240
Finding Common Syntax Errors	242
Scoping errors	242
Typing errors	243
Incorrect data types	244
Finding Common Semantic Errors	245
Infinite loops	245
Off by one	246
Strategies for Debugging	248
Turning sections on and off	248
Testing sample data	251
Adding output messages	251
Walking Away	253

CHAPTER 14: Creating a Webpage	255
Getting Set Up	255
Creating a Basic Webpage Layout	261
The skeleton: HTML basics	262
The aesthetics: CSS	265
Getting Fancy with Color and Graphics	272
Adding color to your page	272
Introducing graphics	276
Adding Hyperlinks	278
Going Interactive with JavaScript	280
Adding buttons	280
Changing your page with buttons	282
Combining HTML, CSS, and JavaScript	283
CHAPTER 15: Building a Mobile Game	289
Getting Started with MIT App Inventor	289
Community and support within MIT App Inventor	291
The layout of MIT App Inventor	292
Using an Emulator versus a Real Device	294
Using the Android Emulator	294
Using a real Android device	295
Testing on the emulator and Android device	295
Designing Mobile Apps	302
Adding the Components in Design View	303
Coding Your Mobile App	306
Getting your puppy moving	306
Setting up your start screen and variables	308
Coding random placement of items	309
Coding collision with items	311
Levels, timers, and final score	312
Distributing Your Apps	315
CHAPTER 16: Programming Simple Electronics	317
Gathering Your Hardware	317
The micro:bit board	318
Buying the board and components	318
Accessing the Software	320
Navigating the interface	320
Writing and using a program	321
Don't Wake Baby Gadget	324
Flowcharting the program	324
Writing the code	326
Downloading code to the micro:bit	333
Connecting hardware components	333
Testing the device	334
Trying Wacky and Fun Variations	335

PART 5: THE PART OF TENS	337
CHAPTER 17: Ten Do's and Don'ts for Selecting a Kids Coding Curriculum	339
DO Find the Right Entry Level	340
Getting started in elementary grades.....	340
Getting started in the middle grades	341
Getting started in high school grades.....	341
DON'T Assume Cost Equals Quality	343
DO Balance Lessons with Free Exploration.....	344
DON'T Instantly Dismiss Teaching Languages	344
DO Consult CSTA for Guidance	346
DON'T Buy "Coding" Toys for Babies	346
DO Emphasize the Soft Skills.....	346
DON'T Let Kids Get Stuck in a Loop	347
DO Present the Bigger Picture.....	347
DON'T Stereotype Coders	347
CHAPTER 18: Ten Ways to Keep the Coding Learning Going	349
Unplugged	349
Research Pioneers of Computing	350
Go Lateral from Code	351
Language Tracking	351
Smart Home Projects	352
Include Outside Passions.....	352
Open-Source Projects.....	353
Group Projects.....	354
Community Support	354
Portfolios	355
INDEX	357

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



TIP

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.



REMEMBER

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



WARNING

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



TECHNICAL
STUFF

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.

- » 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 appreciating 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 environments 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