

3

Try Your Hand at Coding

Now that you have some sense of what coding, programming, and computer science (CS) are about, let's jump into some experiential learning. In this chapter, we encourage you to hop into some challenges, ignoring (for now) the practical application to teaching and learning in your classroom. Think of this instead as an open call, an opportunity to join the cast of this production called Computer Science!

Computer science education cannot make anybody an expert programmer any more than studying brushes and pigment can make somebody an expert painter.

—Eric Raymond, software developer and author

As with almost any subject, the best way to learn CS deeply is to prepare to teach it. In many cases, the first step toward feeling comfortable teaching something new is to have some positive personal experiences with it. In our opinion, the best way to ensure a positive personal experience with CS is to start at the very beginning.

TIME WELL SPENT

In the remainder of this chapter, we walk you through a few exercises that, as they progress, increase in both complexity and ultimate reward. You may feel tempted to take a pass on preliminary activities, as they may seem trivial or overly simple. Be advised that these opportunities are worth your time. Just like a pre-workout stretch helps you stay limber and avoid injury, these exercises get you ready for learning in the chapters ahead. Additionally, it's quite likely that most of your students will be starting this adventure from scratch, so going through this process yourself will help you be an empathetic facilitator of their experience.

Because we respect your time, we considered a great assortment of tutorials before distilling our set down to three warm-ups and two exercises that will prepare you best for the chapters that follow.

To keep tabs on your thoughts, feelings, concerns, and questions, we urge you to grab a notebook or a digital tablet. Take note of your experience with



each activity and write down questions as they come up. If those questions aren't answered in the remainder of this book, then use our companion site, www.corwinpress.com/companion/codinginclass or Facebook page ([facebook.com/groups/CodingInClass](https://www.facebook.com/groups/CodingInClass)) to get them answered. If you are involved in a book study, share your notes about your experiences with your fellow readers, too. Keeping this experience collaborative is a big part of the fun.

PAIR PROGRAMMING



FIGURE 3.1 Key Strategy

© Virtaa/iStock photos

If you really want to make an impact as you go through the following exercises, we recommend pair programming (Figure 3.1). Pair programming is a proven method for both enhancing learning and writing better code. Sometimes called “peer programming,” it is the act of coding with another person by your side. In schools, this happens by seating two students at the same machine and designating one person as the “driver” and the other as the “navigator.” The driver works the mouse and keyboard—typing and maneuvering through the project—while the navigator pays attention to the big picture and makes sure that the code seems logical. The pair works like this for a short time (often by time limit, number of problems solved, or some other measurable mark); then they switch places.

It's easy to see the benefits of pair programming in an educational setting. First, pair programming helps teachers to accommodate an entire classroom with half as many machines. Also, students in both roles are thinking aloud, which is a metacognitive strategy for evaluating and improving reasoning. As an added bonus, the fact that each group has its own built-in sounding board cuts down on the number of hands being raised for teacher assistance (Williams, Kessler, Cunningham, & Jeffries, 2000). This means that students are also less likely to quit in frustration and are more likely to have fewer bugs in their resulting program (Williams & Upchurch, 2001).

The data surrounding pair programming is so strong that some companies have adopted it professionally. Often incorporated as part of an “extreme programming” mentality, studies show that these programming pairs turn out code of significantly higher quality in nearly the same amount of time as if each of the individuals had been working on his or her own. Besides, it's nice to have a buddy to bounce your ideas off of, and this method offers an environment for encouragement when a task seems overly daunting.

For all of the previously mentioned reasons, as well as the desire to spread CS to every classroom in the nation, we suggest that you go through the following exercises with a partner. If you're unable to find someone who can physically sit with you at your machine, arrange a Google Hangout,* and share your screen as you go. It's more than twice as nice to have another person to talk to and think aloud with as you progress through these activities.


Alone we can do so little, together we can do so much.


—Helen Keller, author and activist


Now that you're prepared, it's time to jump in. As with any good workout, we'll start you with some warm-ups and stretches before we move to the main exercises. Give each a thorough try. As you go, take notes about your experiences, and consider the reflection prompt that follows each warm-up or exercise before you move from one to the next.



TEACHER WARM-UPS AND EXERCISES

Set aside a couple of hours to dig in to these warm-ups and exercises with a partner. Start at the beginning of the list, and move through sequentially, devoting 15–30 minutes to each. Every activity helps you build skills, confidence and familiarity that you will draw on as you read the rest of the book.


| Warm-ups | |
|---|--|
| <p>Warm-up 1: Magic Pen—Learn to think differently (Play for 15–20 minutes)</p> |  http://goo.gl/Y8puxq |
| <p>Description: This flash-based physics game challenges a user to solve puzzles with only their magical pen and a library of shapes. Use this application to contemplate multiple solutions to difficult problems. The solutions are limited only by your imagination.</p> | |
| <p>Computer Science Tie-in: Break up the long vignettes into shorter puzzles using decomposition. See if you can find similarities between obstacles using pattern matching; then repurpose a previous solution with new details by abstracting out differences. Be persistent!</p> | |
| <p>Journal Questions</p> <ul style="list-style-type: none"> • How did you feel at the start of Magic Pen compared to after a few levels? • When you hit your first point of frustration, were you tempted to give up? • What kind of pep talk did you give yourself to persist a little longer? | |
| <p>http://media.abcya.com/games/magic_pen/flash/magic_pen.swf</p> | |

| | |
|--|--|
| <p>Warm-up 2: Play Auditorium—Persistence and debugging (Play for 15–20 minutes)</p> |  http://goo.gl/HQQfQD |
| <p>Description: With beautiful streams of light and peaceful harmonies tinkling as you play, you will hardly realize how fast your pulse races as the level of difficulty increases. Every puzzle has a solution, but they are far from obvious.</p> | |
| <p>Computer Science Tie-in: This is the ultimate test of persistence! Hang in there as you pass from challenge to challenge, debugging along the way. After each move, interpret the results to figure out what to do next. Are you getting closer to or further away from a solution?</p> | |
| <p>Journal Questions</p> <ul style="list-style-type: none"> • Was Auditorium all trial and error, or were you able to read the game's clues? • At what points did you find yourself stopping to think and plan before proceeding? • In what ways was it helpful to have another person go through the levels with you? • Did “thinking aloud” help you combine your reasoning to get to the correct solution? | |
| <p>http://www.cipherprime.com/games/auditorium</p> | |

| | |
|---|--|
| <p>Warm-up 3: Lightbot—Using the computer to program (Play for 15–20 minutes)</p> |  http://goo.gl/54dWWK |
| <p>Description: You'll be coding before you even know what's happening with this fun and friendly online game. Use instruction blocks such as a spring to jump and arrow to move forward as you navigate your lightbot among the squares to get to specific goal spaces.</p> | |
| <p>Computer Science Tie-in: This application is intended to help you get used to dragging blocks to control the actions of your character and to learn how the process of calling functions enhances a program.</p> | |
| <p>Journal Questions</p> <ul style="list-style-type: none"> • Do you see how your activities relate to programming? • In what ways did intuition help you progress? • How did talking through the problems and even gesturing contribute to your progress through the levels? | |
| <p>http://lightbot.com</p> | |

| | |
|--|---|
| <p>Exercise 1: CS Fundamentals Course 1—Vocabulary and concepts (Play for 30–45 minutes)</p> |  |
| <p>Description: This step-by-step tutorial is a true introduction to the basics of CS. You will learn about algorithms, debugging, persistence, loops, and events in this bona fide CS curriculum! Try to go as far as you can with this introductory course.</p> <p><i>(You can skip the “unplugged” lessons, but take note of them for later.)</i></p> | <p>http://goo.gl/OVGtsA</p> |
| <p>Computer Science Tie-in: Use this application to get familiar with CS vocabulary and concepts, and learn how to combine foundational elements like conditionals and events to solve tricky problems.</p> | |
| <p>Journal Questions</p> <ul style="list-style-type: none"> • <i>You’re programming! How does it feel?</i> • <i>Were you able to get the hang of the block-based programming style?</i> • <i>Did you get to experience loops and events?</i> • <i>How might this be helpful when learning other concepts, like functions and variables?</i> | |
| <p>http://studio.code.org/s/course1</p> | |
| <p>Exercise 2: Javascript.com by Code School—Practically programming (Play for 30–45 minutes)</p> |  |
| <p>Description: A smooth walk into text-based coding, this tutorial is a precursor to Code School’s JavaScript Road Trip. Follow the directions to learn the basics of JavaScript, a powerful and flexible language for HTML and the web.</p> | <p>http://goo.gl/RhMF3m</p> |
| <p>Computer Science Tie-in: Experience the beauty of typing lines of code into the console and watching your computer respond to your commands. This is the last step before preparing to write entry-level apps of your own.</p> | |
| <p>Journal Questions</p> <ul style="list-style-type: none"> • <i>What was it like to control the computer with a language that is currently popular in the industry?</i> • <i>Were you able to start picking up clues about some of the rules (like putting quotes around strings of characters or using semicolons at the end of statements)?</i> • <i>Can you imagine writing a program on your own, from the very beginning, or are you more inclined at this stage to continue building from a program that already has a framework?</i> | |
| <p>http://www.javascript.com</p> | |

Summary

| | |
|--|--|
| In Summary: Journal and Share —Post to Facebook group |  https://goo.gl/h6P7NC |
| Description: Summary reflection of warm-ups and exercises | |
| Journal Questions <ul style="list-style-type: none">• <i>What is your overall impression of “programming?”</i>• <i>Were you able to think of these tasks as puzzles and challenges, or did they feel like homework?</i>• <i>Which activity made you the most excited, and why?</i>• <i>Which really tested you, and how did you feel about feeling uncertain at times?</i>• <i>What would you tell someone else about your first experiences?</i> | |
| https://www.facebook.com/groups/CodingInClass | |

What Have I Done and What Do I Do Next?

This chapter has taken you on a long journey from the place of a lone educator exploring the possibility of incorporating CS into the classroom to that of an entry-level programmer! Now that you’ve had a taste, take a moment to pat yourself on the back. Celebrate a bit, then dive into the next chapter, where we start imagining what CS could look like as part of a well-rounded classroom curriculum. Lights, camera, action!

Getting Started in the Classroom

As a teacher, you don't have to look far to find concerns when it comes to integrating computer science into everyday instruction. You or your colleagues may be concerned about fitting one more piece of curriculum into an already crowded program. Parents worry about adding more “screen time” to a child's routine. School administrators aren't certain computer science is worth the cost of retraining teachers.

The trick, and moral responsibility, is to integrate computer science activities in a way that elevates the “pros” while diminishing the “cons” so students get the experiences they need.

When adding computer science to your plan, you can keep many issues under control with a simple gut check. Do students find all the computational activities interesting? If not, some may be overused. Do the activities enhance a science investigation or creative process? Creating a computer model or simulation can help your students to see or experience information in a way that is otherwise impossible, and the possibilities for digital art are endless. If your answers all fall on the side of enhanced student experience, then you're on the right track!

START LOW-TECH

One way computer science can be integrated into the classroom without additional screen time is through “unplugged” activities. Unplugged lessons are computer science activities that do not require digital devices or the internet. Consider them the “live theater” of CS! Often, they involve arts, crafts, games, and movement to get across vocabulary and concepts as complex and diverse as you might find in an entry-level college class. Showing is easier than telling, so give one of our “unplugged” lessons (like “Algorithms and Automation—A Compliment Generator” on page XXX), and peruse the companion site for links to some of our favorites.

ENCOURAGE MOVEMENT

Part of being a responsible CS educator is teaching your students how to be responsible CS learners. This means students need to think about taking care of their bodies, their minds, and their environment.

When it comes to the body, the use of technology can take its toll. Sitting at a computer for long periods is not good for anyone's health. Encourage kids to get up and walk around often, especially when programming on a machine. Allowing students to walk freely about the room to study what others are doing might be frowned on in math class, but in computer science, active collaboration is an effective learning technique.

Computer activity should take place in small chunks of time, ideally for no longer than you would expect a student to be able to sit and read on their own without interruption. Why? Well, when we are out in the world, humans usually look around in three-dimensional (3D) space with nearly 180 degrees of range top to bottom, and side to side. When on a computer, we focus only on a 2D space with a range of about 30 degrees top to bottom and 40 to 50 degrees side to side ("Monitor Height and Position Guidelines," 2008). Spending too much time in this limited, high-contrast atmosphere can cause eye strain and muscle soreness. With this in mind, encourage students to follow the 20/20/20 Rule: Every 20 minutes, get up and stretch while looking at least 20 feet away for at least 20 seconds. Set a timer if you need to.

20/20/20/ Rule: Every 20 minutes, get up and stretch while looking at least 20 feet away for at least 20 seconds.

These breaks are a great opportunity for conversation, too. If you want to give students' full bodies a break, maybe some dance moves are in order! The Tut, Clap, Jive activity that we refer to on page XX would be fun day after day.

Issues of sedentary living extend beyond school, of course. Teach students that for each hour of screen time they have in a day, they should spend the same amount of time outside playing, engaging in sports, or just going for a restorative stroll. If they begin to hear this encouragement early, being active has a much better chance of becoming a habit than if they only hear it later in life.

When it comes to technology, it's your job to help students manage their own well-being. Now that we've covered the physical bit, it's time to talk about mental steadfastness as it relates to the digital world.

FOSTER CRITICAL CONSUMPTION

A recent report from the United Kingdom (“Children and Parents: Media Use and Attitudes Report,” 2015) shows that students are increasingly believing everything they read on the internet. This problem is probably compounded by the likelihood that their parents believe everything *they* read on the internet, too, and pass the information along. To exacerbate things further, intelligent search engines (like Google) have tuned their algorithms to the point where they offer “top results” based on known preferences (Herlocker, Dieterich, Forbes, & Maritz, 2012), so the “best” results for any search you create will tend to be consistent with whatever viewpoint you already have (White, 2013).

In the same way, social media sites like Facebook saturate your feed with posts from friends that you interact with the most, making it all but certain that the majority of viewpoints that you’re exposed to will continue to fuel your existing beliefs.

This is a great time for solidarity, but an awful time for the open-minded exploration of differing points of view. For that reason, every student should receive a thorough education on the credibility of the web. While not explicitly computer science, the care and proper use of search engines is important to CS education, and therefore, it is important to discuss in this book.

It helps for students to recognize that, like movies, web pages can seem authentic and still be entirely fictitious. Even after this message has been absorbed, there is power and authority in published words from other sources. It is worth taking your students through some exercises where they see how ludicrous even some of the most believable pages are. Teach them how to “check their work” in the same way that they might for math class. Have them take a “fact” and trace it back somewhere other than Wikipedia. Is one of the first results on the page from Snopes.com?

Truly educated citizens need to understand where their facts are coming from, and while we gave you a small taste of the need to learn about our sources of information, it would require another book entirely to do the subject justice. Please see our companion website for more details around lessons in this subject.*

People are very gullible. They’ll believe anything they see in print.

—E. B. White, *Charlotte’s Web*

PROTECT PRIVACY AND PREVENT CYBERBULLYING

It is said that “familiarity breeds contempt,” but apparently anonymity breeds disregard, criticism, and hostility. A study by the University of

Houston found that anonymous users were nearly twice as likely to post uncivil comments than users who were somehow required to identify themselves (53.3% to 28.8%, respectively; Santana, 2013). This explains a lot of the horrid behavior on the internet.

Digital divide:
Inequalities perpetuated by disparate access to computers and the internet.

Students today have it hard enough straddling the **digital divide** with the pressure to be carrying the newest, best device. When you add anonymous internet interactions to the mix, it creates the stuff of teenage nightmares. Since roughly 91% of all teens access the internet from a mobile device, and nearly 71% of all teens frequent more than one social media site (Lenhart, 2015), it's almost certain that each of them will run into haters at some point.

If not monitored properly, the internet can be turned into a psychological weapon ("The Top Six Unforgettable Cyberbullying Cases," 2013) full of venom and danger. So what is an educator to do when encouraging students to browse and share on a regular basis?

Up to Age 13: Monitor and Protect

Most students younger than age 13 are not allowed to have their own accounts on social media in the United States. If you are using blogs or online learning sites in your classroom, make sure that you are in control of setting up each account. Educational sites for children (like Codecademy, Code.org, and Edublogs) will generally have bulk sign-up pages to associate students with your teacher account without sharing too much personal information, such as student IDs or email addresses.

If you are using sites that require individual emails for students, try creating a single Gmail address for each class; then append numbers to the end of it when signing up each student. For instance, if we made the Gmail address k!k!Class@gmail.com, then we could use k!k!Class+1@gmail.com for the first student, k!k!Class+2@gmail.com for the second student, and so on. Every message that came through to any of those accounts would then be delivered to you at the main k!k!Class@gmail.com account.

Age 13 and Older: Trust but Verify

Teenagers and the internet are a tumultuous combination. Teens are old enough to be treated with autonomy and trust, but they are still testing boundaries and challenging consequences. As an educator, how do you strike a balance?

At this age, we recommend that you work with their desire to express themselves and encourage appropriate use of social media in study and as

a portfolio medium for their accomplishments. Be open about the pitfalls of posting on a forum that allows comments from the general public, and encourage students to protect their privacy by allowing only personal friends to access their profiles. Round the topic out by giving a name to cyberbullying and condemning it outright, with zero tolerance and well-defined ramifications. Follow up the conversation by describing what to do when a student believes that they have been the target of a bully online (Cyberbullying Research Center, n.d.).

No matter what age your students are, talk to them about being good digital citizens and respecting others, even when their identity is hidden. Remind students that the true measure of one's character is what they do when no one is looking. Challenge them to defend their own accounts and protect those of others. If your students are prepared to make good choices, then when the time comes to act, they just might do the right thing.

If you aren't already steeped in your district's acceptable use practices (AUPs), look them up and respond accordingly. If you'd like a deeper dive into digital safety and citizenship, read *Digital Citizenship in Schools*, by Mike Ribble (ISTE, 2015).

ACHIEVE ACCESS

Teachers often cite two hurdles when it comes to incorporating computer science into their classroom: time and budget restrictions. Both of these can be addressed by selecting the right curriculum for your classroom.

Computer science can be introduced into multiple subjects even without having entire classes devoted to the subject. The truth is, computer science is the understanding of computing as a tool, and that tool can be used in service to any subject. At young ages, Scratch gives you a tool to bring CS into music (Heines, Ruthmann, Greher, & Maloney, 2012), CS Fundamentals allows you to use computer science in art ("Computer Science Fundamentals for Elementary School," 2015), and Tynker brings computer science into math class ("Programming = Better Math Skills + Fun," 2014).

For middle school and beyond, entire organizations are dedicated to interdisciplinary computing, including Bootstrap (math) and Project GUTS (science), which we point to in Chapter 16.

What's more, when students become capable of understanding text-based programming languages, their world opens up with opportunities for CS integration. Imagine students going deeper with projects by adding formulas to spreadsheets, adding interactivity to maps, and creating graphical representations from data.

The cost of equipping a school is nominal for most of the programs listed earlier. Many of the curriculum packages mentioned are free, and some providers even offer free professional development opportunities for educators. Functional equipment becomes the next consideration, but these costs can be mitigated through pair programming (because half as many machines are needed), more agile, on-demand access to school technology (as schools dismantle computer labs and supply more computers to classrooms), and adoption of BYOD (“bring your own device”) policies.

Programming environment: A software workspace made for creating code.

For the cases where classrooms are unable to procure hardware capable of running up-to-date browsers, unplugged CS lessons are the next best option. Many, like Conditionals with Cards* or For Loop Fun* use only paper, playing cards, and/or dice.

Software patch: A piece of software designed to update or fix a computer program.

Suppose, however, that you are one of the lucky ones, and you have found the time and the hardware to implement a full-scale computer science program. You may worry that it’s like the old days when coding meant installing complicated **programming environments** and dozens of **software patches**, but have no fear. Today, many great entry-level programming environments are browser-based and operate online, allowing your students to receive the benefits of an introductory CS education through a handful of quality websites.

BANISH ANXIETY

In reality, we have found that the biggest barrier to providing students with a solid introduction to computer science is teacher anxiety.

Worry not, brave instructor, because the most important thing that you can teach in CS is how to learn, and you can model this as you learn alongside your students. Computer science changes fast. A programming language introduced to students their first year of high school might be outdated by the time they graduate. Even the most learned computer science professor doesn’t have enough CS knowledge to answer every stray question from an elementary student. For this reason, we ask that you become comfortable saying, “Let’s find out together.”

Adopt the motto, “Everyone a teacher, everyone a learner.” Students solidify their learning when they teach someone else (Brooks & Brooks, 1993), so let them end a lesson by teaching you and their classmates what they discovered. This is a great opportunity to challenge students to put what they’ve learned—and how they learned it—into words. Thinking through their actions not only helps them accomplish the day’s task but it exposes

the processes of learning something new; a skill that can transfer to every learning challenge they encounter for the rest of their lives.

If you are used to dispensing knowledge and controlling most aspects of the learning experience, give yourself permission to let go—even if for only one group of kids around one topic. We promise the experience will be both enlightening and enriching for everyone involved.

Have no fear of perfection—you'll never reach it.

—Salvador Dali, artist

Copyright Corwin 2017