frontiers | Frontiers in Water

METHODS
published: 22 June 2022
doi: 10.3389/frwa.2022.875732

Best Management Practices for Teaching Hydrologic Coding in Physical, Hybrid, and Virtual Classrooms

Christa A. Kelleher[1,2]*, *John P. Gannon*[3], *C. Nathan Jones*[4] and *Şule Aksoy*[5,6]

[1] Department of Civil and Environmental Engineering, Lafayette College, Easton, PA, United States, [2] Department of Earth and Environmental Sciences, Syracuse University, Syracuse, NY, United States, [3] Department of Forest Resources and Environmental Conservation, Virginia Tech, Blacksburg, VA, United States, [4] Department of Biological Sciences, University of Alabama, Tuscaloosa, AL, United States, [5] Department of Science Teaching, Syracuse University, Syracuse, NY, United States, [6] The Graduate Center, The City University of New York (CUNY), New York, NY, United States

OPEN ACCESS

Edited by:
Anne Jefferson, Kent State University, United States

Reviewed by:
Nichoas Kinar, University of Saskatchewan, Canada
Jordyn Wolfand, University of Portland, United States
Dominick Ciruzzi, College of William & Mary, United States

***Correspondence:**
Christa A. Kelleher
kellehec@lafayette.edu

Specialty section:
This article was submitted to Water and Hydrocomplexity, a section of the journal Frontiers in Water

Received: *14 February 2022*
Accepted: *23 May 2022*
Published: *22 June 2022*

Citation:
Kelleher CA, Gannon JP, Jones CN and Aksoy Ş (2022) Best Management Practices for Teaching Hydrologic Coding in Physical, Hybrid, and Virtual Classrooms. Front. Water 4:875732. doi: 10.3389/frwa.2022.875732

As the field of hydrologic sciences continues to advance, there is an increasing need to develop a workforce with tools to curate, manage, and analyze large datasets. As such, undergraduate and graduate curricula are beginning to regularly incorporate scientific programing in the classroom. However, there are several key challenges to successfully incorporating scientific programming into a hydrology course or curriculum, such as meeting disciplinary outcomes alongside teaching students to code, equity issues with access to computing power, and effective classroom management. While these challenges were exacerbated by the global pandemic, shifting to online and hybrid learning formats provided an opportunity to explore and re-evaluate the way we facilitated our hydrology courses and integrated coding exercises and learning. In this article, we reflect on these experiences in three very different hydrology courses (e.g., courses housed in geoscience/engineering, environmental science, and biology programs) with an eye toward identifying successes and opportunities for improvement. We explore this by presenting ten best management practices (BMPs), representing a series of recommendations we have for teaching a virtual, hybrid, or in-person hydrology course that incorporates coding. While all recommendations provided can be applied to many programming languages, the focus of the paper (given the expertise of the authors) is on R. Our BMPs focus on technological facilitation, managing the virtual classroom, and instructional resources, with lessons learned that are applicable to in-person instruction. We also summarize the ways that the authors of this article integrate coding into our coursework to serve as a framework for prepping new courses or those revising existing hydrologic coursework. Above all, we hope these series of recommendations will evolve as hydrology courses continue to emphasize computational skills alongside disciplinary learning.

Keywords: coding education, hydrology, hydrogeology, computational thinking, STEM education and learning

INTRODUCTION

The field of hydrologic science—as well as science, technology, engineering, and mathematics (STEM) fields—is built on numerical inquiry. As hydrologists, we use data to interrogate research questions, to support decision-making, to benchmark variations and change, and to deliver new design solutions. As computing capabilities have continued to advance, and the volume and variety of the data we work with has continued to expand, many professional hydrologists and hydrogeologists are turning to scientific programming languages, including R, Python, and MATLAB, to complete daily tasks.

Scientific programming and closely associated skills are prized within the STEM workforce. A recent US federal report emphasizes that developing a national STEM workforce strategy goes hand-in-hand with promoting an understanding of the basics of computing and data science through research-based pedagogical practices (National Academies of Sciences, 2016). Indeed, work by Carnevale et al. (2011) identified the importance of knowledge of computers and electronics, an overarching knowledge domain that includes computer applications and coding, as 1 of 10 core knowledge domains most closely associated with STEM occupations. By their estimates, computer and electronics knowledge is not only crucial to a very high percentage of STEM occupations, but also represents a transferable skill beyond STEM occupations (Carnevale et al., 2011). Recent annual surveys indicate that more than 50% of superiors working closely with college graduates ranked skill sets associated with "complex problem solving", "critical thinking", the "ability to analyze and interpret data", and the "ability to work with numbers and statistics" as "very important" for new graduates (Finley, 2021).

Within the educational literature, the process of learning to write code most commonly aligns with developing abilities in computational thinking (Wing, 2006). Computational thinking is often described as the process of defining a problem and associated solutions such that either a human or machine (or both) can execute the proposed solutions (Wing, 2006). While no commonly accepted definition for computational thinking exists, the definitions used across the literature emphasize abstraction and automation (Lyon and Magana, 2020). Computational thinking and, more broadly, computational knowledge, are often developed within computer science (CS) courses as well as in courses that teach programming in disciplines beyond computer sciences, with the educational literature drawing distinction between the two. Around the world, countries are implementing the inclusion of computational thinking, digital literacy, and computer programming across the curriculum, and at the K12 and undergraduate level (National Academies of Sciences, 2016; The Royal Society, 2017; Valente and de Almeida, 2020; Nesen et al., 2021). Though the educational literature on computational thinking within programming courses (i.e., those outside of computer science departments) as well as the existence of publicly accessible coding exercises across STEM fields is expanding (Jacobs et al., 2016; Yan, 2017; Lin et al., 2019), understanding of best practices is still nascent, particularly at the disciplinary level. Overall, there is limited research on teaching and learning to code; and there is a general lack of educational materials to support teaching scientific coding methods (Medeiros et al., 2019). For this reason, core practices and publicly available repositories of teaching resources are still lacking.

The onset and continued evolution of the COVID19 pandemic has fundamentally changed the way we deliver content to students, and how we, the authors of this article, specifically taught scientific programming. In particular, the pandemic brought into sharp focus how we not only teach about computing technology, but how we most effectively use computing technology to accomplish this. Likewise, it forced us to consider what elements of our delivery were most effective in a virtual or hybrid classroom. Considering our approaches to teaching coding, it became even more important to minimize troubleshooting problems beyond just those with code (software versions, wrong directories, and more). Finally, it gave us the opportunity to reflect on and revise our courses and associated expectations from the lens of equity and inclusion within the classroom. This article represents ten best management practices (BMP), a double entendre given that BMPs are also used as conservation practices in hydrologic science, that we have arrived at for delivering a coding course at the undergraduate level. These BMPs represent what we view as core practices, in that they are constantly evolving through collaboration and feedback from learners and practitioners. Our hope is that while these recommendations were formulated in the context of virtual and hybrid teaching, they will improve student learning outcomes upon returning to in-person instruction.

WHAT DO WE MEAN BY "CODING"?

In this article, we are focused on approaches that we use to teach others (particularly undergraduate and graduate students) how to write code. Throughout the manuscript we primarily use the term "coding" instead of "programming". As highlighted by Corradini et al. (2018) the term "coding" may be preferred to "programming" given the many publicized initiatives that include "coding" in the title, broad media use of the term, and that, to put it simply, the term "coding" may sound more interesting or exciting than "programming". By "coding", we are referring to the process of students building skills in writing one or more lines of code to perform analysis, including computations, generating visualizations, or writing programs or functions (e.g., multiple lines of code that build to achieve some output; Van Merrienboer and Krammer, 1987). In non-CS disciplines, this often requires combining learning discipline specific knowledge alongside the commands and syntax of a given programming language (Van Merrienboer and Krammer, 1987). The authors primarily use the R coding language. Thus, most of the examples are given in R, but we note that the principles introduced in this text can be translated to any other programming language such as MATLAB or Python.

COURSES

Our reflections in this article are shaped by our experiences in developing and teaching three different courses that incorporate coding as part of course learning objectives and content. We briefly outline these three courses below.

Course 1—Physical Hydrology

A mixed undergraduate-graduate course aimed at introducing students to physical hydrology concepts. The course size was 18 students, with backgrounds split between the Earth Sciences and Civil Engineering and met twice a week for 75 min. The course was taught at Syracuse University (Syracuse, NY, USA). During the pandemic, the course was taught in the spring 2021 semester, though was offered in prior semesters as well. The course operated in hybrid mode for lectures (1 day a week) and in an online environment for all but the last few weeks for coding examples (1 day a week). Students were introduced to R coding to quantify hydrological processes, perform hydrologic analyses and hydrological modeling, and to compare watershed responses in variable environments.

Course 2—Hydroinformatics

A mixed undergraduate-graduate course taught at Virginia Tech (Blacksburg, VA, USA) aimed at introducing students to common introductory to moderately complex hydrologic analysis using coding. During the pandemic, the course was taught in 2021 during the spring semester, though was taught prior to this offering. The course was 14 students, with backgrounds in undergraduate data science and hydrology courses, as well as graduate students with a variety of backgrounds. This course was taught synchronously online twice a week for 75 min. Students were introduced to R over the first 2 weeks of the course, and then each week the class introduced hydrologic analyses and the coding concepts necessary to complete them. Topics include basic statistics, flood and low flow statistics, modeling, and basic geospatial operations like delineating watersheds and stream networks.

Course 3—Ecohydrology

A mixed undergraduate-graduate course aimed at introducing students to the water cycle and ecosystems taught at University of Alabama (Tuscaloosa, AL, USA). Course enrollment included 20 students with backgrounds in environmental science and biology and was taught in fall 2021. Weekly meetings included a 1-h synchronous virtual lecture where students were introduced to a broader topic, 1-h journal article discussion (graduate students only), and 6-h lab. The longer lab period allowed the class to be broken up into smaller groups. The first half of the course focused on components of the water balance (i.e., watershed storage, precipitation, streamflow, and evapotranspiration), and the second half of the course focused on how water interacts with ecosystems (i.e., wetland soils, plant-water interactions, catchment biogeochemistry, and flow-ecology relationships). Labs included a range of activities that included both empirical data collection and data analysis using R.

DISCUSSION

Below we summarize 10 considerations for incorporating coding into hydrology courses. We developed many of these practices through our experiences with pandemic teaching but emphasize that nearly all recommendations are applicable to a virtual, physical, or hybrid classroom environment. For any instructors who are currently teaching coding or considering adapting their disciplinary courses (in hydrology or otherwise) to include coding components, we underscore (as those beyond a novice stage can forget) that learning to write scientific coding is effectively learning a new language and can therefore be daunting at any point in the process (Medeiros et al., 2019). Colloquially, we've collectively encountered many students who have told us they're "just not good at coding". Our answer to these students is to emphasize that coding is just another skill that can be learned through practice, like teaching. In our classrooms, we all sought to foster a growth mindset throughout our courses (McGlynn, 2020), with many of the BMPs below reinforcing this overarching and ever-important tone.

BMP 1: Motivate the Importance and Benefits of Learning to Code Early in the Semester

In our colloquial experience, we have found that starting a course focused on coding off on the right foot requires demonstrating to students (i) why they should care about learning to code and (ii) how this skill will serve them now and into the future. One of the best ways we've found to motivate and excite students about the value of learning to code is to not simply *tell* students that coding is important, but to *show* that coding can save time and effort. In most cases, students have encountered spreadsheet programs (e.g., Microsoft Excel, Google Sheets) and used these programs for data manipulation and visualization. For this reason, we have found that it is impactful to spend class time benchmarking coding exercises against manual approaches combined with spreadsheet programs.

In a hydrology course, one way to show what can be gained by coding with students is by asking them to manually download publicly available streamflow data and transfer it into a spreadsheet program (**Figure 1**). The process to manually download streamflow data from the National Water Information System (NWIS) website can take some time, and even more time at the first instance of students encountering the NWIS dashboard (**Figure 1**). When juxtaposed with the fraction of a second and few lines of code in R (using the dataRetrieval package; (De Cicco et al., 2021), the gains from coding are clear (**Figure 1**). An alternative, complementary approach is to engage students in making a plot in Excel from basic data (such as streamflow or any other dataset that relates to course content). In this process, the students are asked to write a set of instructions for how to make an Excel plot, such that the process can be duplicated by another student, and to indicate or sum the number of "clicks" needed to do so. Again, comparing this process to

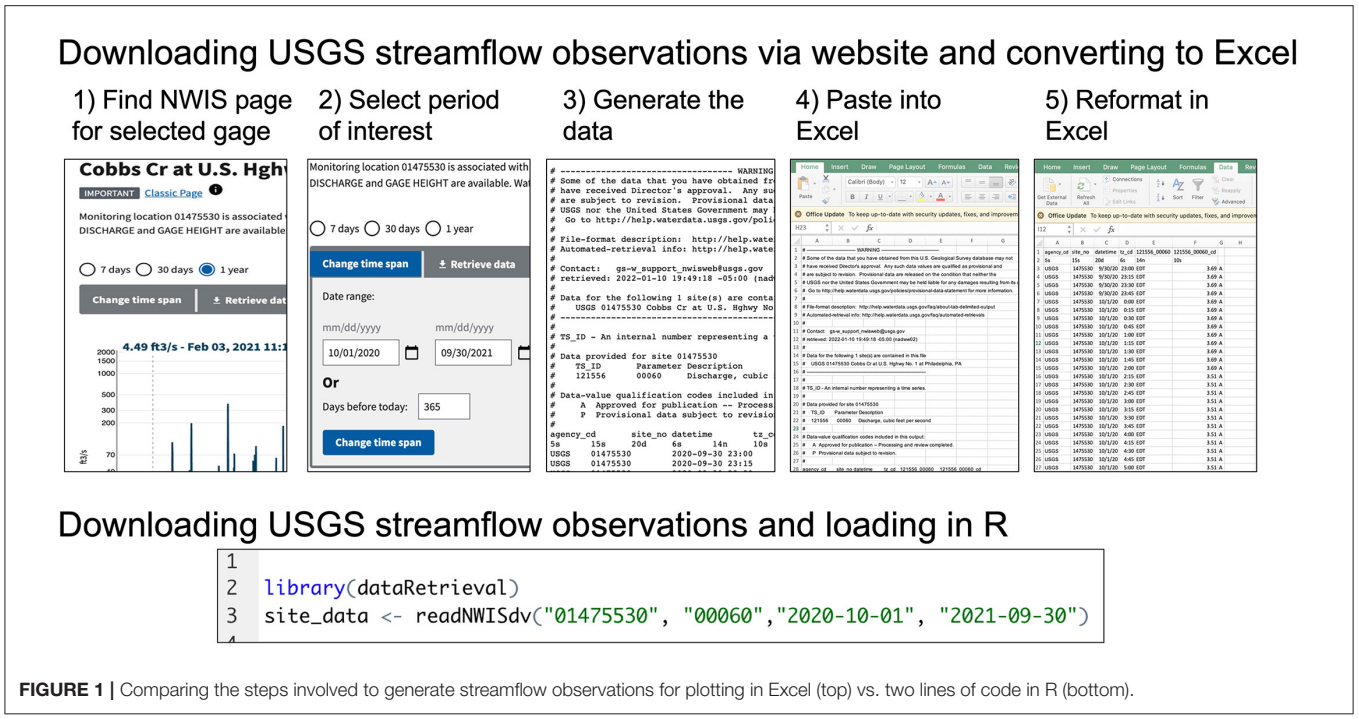

FIGURE 1 | Comparing the steps involved to generate streamflow observations for plotting in Excel (top) vs. two lines of code in R (bottom).

writing a few lines of code to produce a high-quality figure in R, MATLAB, or Python can emphasize the value of learning to code.

Another educational approach that we have used to show students the value of coding is to convey this to them in many formats. Outside of the classroom, one of the first places that educators can make this case to their students is within their syllabi. Within syllabi, it is always worthwhile to address the importance of course content and how it matters to students, both now and in the future; pointing to surveys of the workforce and job growth estimates in certain occupations can all be used to emphasize the value of learning to code in a hydrology course. Graduate student and undergraduate students engaged in research will have the additional benefit of coding serving their future research project needs. These same motivators can also be communicated in the first lecture of the semester.

The educational literature strongly supports motivating the importance of learning to code early in the semester (Carter, 2006; Jacobs et al., 2016; Yan, 2017). In a literature review focused on teaching surrounding introductory coding, Medeiros et al. (2019) found that student motivation was one of the biggest difficulties faced by students while learning to code. Likewise, instilling (and maintaining) student motivation is also a challenge for instructors. More broadly, self-determination theory suggests that intrinsic motivation plays a role in promoting learning in educational settings (National Research Council, 2000; National Academies of Sciences, 2018; Ryan and Deci, 2020). For example, if an instructor provides opportunities for students to take ownership of their work, it could lead to autonomous forms of motivation and enhance learning and engagement. Engaging students in metacognitive talk about what they want to achieve by the end of the course could also promote autonomy. Supporting learning through authentic context, for instance, by providing relevant examples to their daily lives, could motivate students as well. Drawing on students' lived experiences and prior understanding helps facilitate engagement and active learning.

BMP 2: Start Slow, and Remember Not Everyone Is Beginning With the Same Knowledge and Technological Ability

When reflecting collectively on our experiences incorporating coding into our classrooms, it was relatively easy for us to identify common pitfalls we all encountered. To this end, **Figure 2** highlights these common pitfalls that we recommend considering in the context of this best management practice. Why did we encounter these pitfalls? Simply put: it's easy to forget what we didn't know when we ourselves first learned to code. Often referred to as the "curse of knowledge," it is easy for instructors to skip introductory or fundamental concepts that are needed for a given task or skill (Kirschner and Hendrick, 2020). Moreover, students arrive in our classes from a variety of backgrounds (i.e., confirmation of understanding from one student is not necessarily representative of the entire classroom). Addressing students' prior understanding is central to learning and engagement (National Research Council, 2000). Therefore, when designing activities, it is important to remember students are still developing their conceptual models of both hydrological processes and requisite coding skills.

To both support student learning and to encourage instructors to "start slow", we suggest instructors practice "reverse engineering" activities, outlining the steps that it took them to achieve a given end point, and scaffolding these steps into a

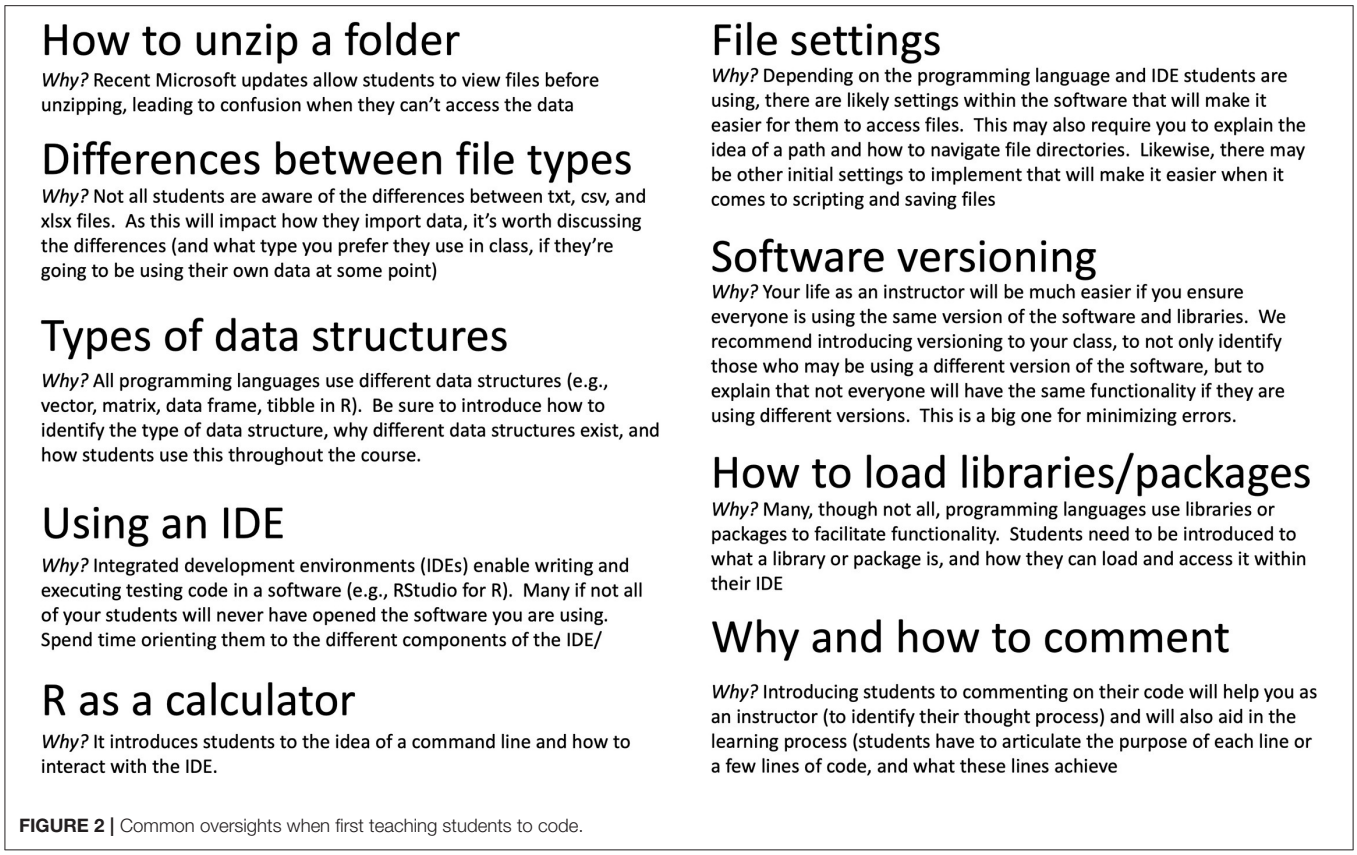

FIGURE 2 | Common oversights when first teaching students to code.

given activity for students. Moreover, we encourage instructors to then have students reverse-engineer their process for each coding exercise, articulating what each step or line of code achieves, and how they know this. Such approaches make student thinking an explicit component of class exercises, and encourage students to deconstruct their approaches, which can enhance their learning (Kirschner and Hendrick, 2020). Furthermore, as discussed previously, consider downloading streamflow data from the USGS National Water Information System (NWIS; https://waterdata.usgs.gov/nwis). This is a fairly common task for hydrology students; and within the R environment, streamflow data can be downloaded with one line of code using the dataRetrieval package (De Cicco et al., 2021). On the surface, it seems like this should be a simple exercise – the instructor should be able to provide an example line of code and move on to more advanced analysis. However, students must understand several concepts before being able to execute this one line of code. These concepts include: (i) What is an integrated development environment (e.g., RStudio), (ii) How do we operate command line software, (iii) What are libraries and how do we use them, (iv) What are data types and data structures, (v) How do we create variables and store/manipulate different data structures, and (vi) What is a function and what is the syntax required to use it. Notably, this list only considers scripting concerns; thus, providing information on streamflow measurements and data may be just as (if not more) important depending on the lessons learning objectives.

To reverse engineer activities, take a step back and consider all the steps necessary to complete the activity. As illustrated by the example of downloading streamflow data, even a simple 2-line task requires a basic understanding of operating within a coding environment. If you have not introduced students to these concepts, it will be necessary to do so. As you are reverse engineering your activity, it may be worthwhile to iteratively revisit and adjust learning objectives to balance hydrologic knowledge with coding skills.

BMP 3: Center Equity and Inclusion From the Beginning

From our perspective, equity and inclusion are the top concerns when teaching coding. If students feel they do not belong or do not have access to your material, programs, or other aspects of the class, the other components of your teaching will be ineffective. Through our experience in the classroom, we have identified three common barriers to running an equitable and inclusive classroom, especially in the context of virtual teaching, and offer specific solutions to each below. These issues are access to high-speed internet, access to a fully functioning computer, and student confidence issues stemming from feelings of imposter syndrome, stereotype threat, or a lack of self-efficacy. We want to note that all authors have experienced all of these challenges in every hybrid or online coding class they have taught. As these challenges are widespread, building your class to anticipate their

existence can save instructors a lot of time while improving the experience for all students.

Internet Access and Quality

It is common for students to have slow or faltering internet connections for a variety of reasons. These include but are not limited to unstable housing, poor quality rural internet, power outages, or simply an overloaded connection with too many users. These issues cannot be fixed, but we can adjust the structure of our classes, so they do not leave these students behind. First, it is imperative to record lecture sessions in an online environment. This benefits students whose internet fails during class, but also, if you make these recordings available to everyone, students who just want to re-engage with lectures to help learn the material. Second, flexible deadlines and/or no penalties for late work can drastically reduce stress for students trying to find a stable internet connection in time to complete work. If instructors are flexible and compassionate, these strategies and lower stake assignments can help students perform better in the course. This also saves instructor time, as it reduces email volume and adjustment of scores/deadlines in your learning management system (LMS).

Functioning Computers

Another common issue, especially when trying to install and run software on student computers, is that student computers are not all of the same quality nor can they achieve the same level of functioning. Even at universities that require students to have a computer with minimum capabilities, by students' third or fourth year, those computers are often in disrepair, and may run slowly and be poorly functioning. Furthermore, students may have netbooks or chromebooks, meaning that such computers are incapable of running the software needed for coding. However, these issues can largely be addressed by offering ways for students to run required software in an internet browser window through cloud-based applications or virtual machines. RStudio can be run in a browser window using services such as binder (https://mybinder.org) and rstudio.cloud (https://rstudio.cloud). Additionally, the computing center at your institution may be able to help you set up a system to run RStudio on a virtual machine or server application. Moreover, many schools are beginning to offer virtual lab computers, where students can run a remote desktop in their browser window. In cases where a student's computer is completely non-functional, being prepared with a laptop that can be loaned out is the best course of action, but this is another benefit of having lectures recorded, as students can work through classroom activities outside of class time on a lab computer.

Imposter Syndrome, Stereotype Threat, and Self-efficacy

Imposter syndrome and stereotype threat are anecdotally the biggest hurdles to students learning to code in our classrooms. In this context, when we use the term "imposter syndrome" we mean students feeling like they don't belong or are destined to do poorly in the class, often manifesting as students sharing with instructors that they "can't code" or "are bad at coding" often when they have not had any coding instruction. Stereotype threat, on the other hand, is the anxiety students feel when they fear they are conforming to a societal stereotype about their social group (e.g., race or gender) and their performance on the subject at hand, and has been shown to negatively affect academic performance (Steele et al., 2002).

It is best to address imposter syndrome and stereotype threat directly. For instance, you can communicate to your students that you designed the class and activities to teach everyone from the beginning, using methods that research has shown enhance learning for everyone. Another approach is to encourage students to focus more on learning goals than performance goals. In the classroom, comparing coding to other skills can contextualize the learning process. For instance, remind students that they wouldn't expect to be able to just jump on a skateboard and effortlessly cruise around, so they shouldn't expect to immediately grasp coding concepts and practices. In both cases the key to improvement is practice.

An additional strategy is to communicate to students that those who put in effort and practice are the ones who learn the material best, not those who are "naturally" good at it. One impactful way to do this is to have students from the previous semester write about their experience in the class: their initial impressions of the course, their approach to the course, and their recommendations to students in the future. Sharing several of these, especially from students who found the material challenging early-on but then did well, can be a powerful tool for pushing back against student fears. Finally, we recommend engaging in the literature on ways to combat stereotype threat and other issues of inclusion in your classroom, as there are many other strategies than the few mentioned here (Killpack and Melón, 2016).

Self-efficacy, introduced by Bandura (1977), describes student beliefs about their own ability to succeed at a given exercise. Beyond coding education, science education research consistently emphasizes the importance of self-efficacy in student persistence and success in science (Pajares, 1996; McBride et al., 2020). Furthermore, higher self-efficacy is often, though not always (McBride et al., 2020), correlated with higher academic performance (Meral et al., 2012; Honicke and Broadbent, 2016; Loo and Choy, 2017) and even greater learning satisfaction in computer-based learning environments (Artino, 2008). As educators, we can support student self-efficacy through learning strategies and effective pedagogy.

Very broadly, one way any instructor can support self-efficacy in their classroom is to use active learning techniques, such as many discussed throughout all BMPs. Ballen et al. (2017) found that using active learning approaches improved academic performance for underrepresented minority students and increased all students' perceptions of their science self-efficacy. One approach we commonly employ in our classrooms to support self-efficacy is collaborative learning. Numerous studies in a variety of disciplines have shown that collaborative, peer-to-peer interactions can enhance self-efficacy (Samiullah, 1995; Fencl and Scheel, 2005; Sidelinger and Booth-Butterfield, 2010; Sollitto et al., 2013; Lewis et al., 2021; Stoeckel and Roehrig, 2021), especially in the context of learning to code

(McDowell et al., 2003; Hanks et al., 2011; Salleh et al., 2011; Dirzyte et al., 2021). While peer-to-peer learning can be challenging to accomplish virtually, the use of breakout rooms can be one way to facilitate these types of interactions. In our virtual, hybrid, and in-person classrooms, we have used peer-to-peer interactions (often groups of two or three) during in-class activities, to compare answers for short quizzes (see Section Low Stakes and No Stakes Assessments), and as a part of two-stage exams (see Section Two-Stage Exams) all in support of self-efficacy and learning. Another approach shown to increase self-efficacy is to have students map out their approach to a given problem or activity, and then track their progression and plans for next steps as they proceed (Schunk and Pajares, 2002). In the context of coding education, work by Govender et al. (2014) employed this approach by introducing students to a framework for problem solving that they can use when working through a coding exercise. In such a framework, students are taught to approach each problem holistically, with coding being one piece of the framework to arrive at a solution or end point (Govender et al., 2014). Finally, McBride et al. (2020) showed that accessible, inclusive, and student-centered practices increased students of color and international students' self-efficacy. Though we touch on effective teaching practices later in this article, it is equally important to mention this topic at this point in the text, given such practices are deeply connected with equity and inclusion.

BMP 4: Do Live Coding

All three authors have anecdotally found live coding to be an effective method of teaching students to write code. This anecdotal observation is bolstered by the coding educational literature, which indicates live coding is commonly seen as a "best practice" across different disciplines (Brown and Wilson, 2018; Selvaraj et al., 2021) as it directly supports active learning (Shannon and Summet, 2015) and exposes students to the process that the instructor uses to write code, allowing them to see many components of coding, including debugging and commenting, in action (Rubin, 2013; Raj et al., 2018, 2020). We describe live coding as the practice of an instructor typing out their code out as they explain what the code is doing and how the statements are constructed, with students following along on their own computers, though other definitions exist (Selvaraj et al., 2021). This challenging method of instruction has several advantages, but also comes with pitfalls. Other potential methods to teach coding include a flipped classroom approach and the use of slides or board notes typical of other topics of instruction. In the flipped classroom example, students complete interactive tutorials or watch lecture videos outside of class and work on activities during class time. We found this an especially helpful method, paired with live coding, when introducing introductory topics, as it offers students more guided practice using general introductory tutorials (see more of these in Section BMP 6: Know What Resources Are Available to You). However, we found that as we progressed to more discipline-specific topics, live coding was more advantageous, as it emphasizes in-the-moment problem solving and troubleshooting (Raj et al., 2018, 2020), as well as a view for students into how instructors write and construct code (Selvaraj et al., 2021). In this section, we outline some of the advantages and strategies for avoiding potential pitfalls, which we have learned through time and practice.

In our experience, the two primary advantages of live coding are that (1) students get immediate feedback when their code does not run due to syntax errors and (2) live coding facilitates weaving additional active learning and experimentation into live coding lessons. More generally, live coding, if done well, can serve as a form of scaffolding, an educational term that describes the support provided to learners by (in this case) instructors and more advanced peers to navigate different tasks (Harland, 2003; Anghileri, 2006; Sharma and Hannafin, 2007). In a live coding exercise, a student who types something incorrectly or has a common syntax error will either get an unexpected result or get an error indicating their code cannot be run. Given an appropriate amount of time and support to fix these errors, they can become valuable learning experiences for the student. In the absence of live coding, the identification of syntax problems and other misunderstandings might not occur until the student is working by themselves on a homework assignment. Additionally, when running a live coding session, it is often easy to let students explore and experiment to broaden their understanding. As instructors we can ask them to tweak the code and observe and explain the results. What happens if you flip the x and y axes in a plot? Can they change a parameter value in a function, and what is the result? These can be simple to execute and powerful for enhancing student learning.

Live coding is a challenging method of instruction and not without risk. In our experience the two most common pitfalls are "losing" students (where students fall behind, can't see your code, or get lost), and accidentally discouraging students from thinking they can learn the material. There are a variety of ways you can prevent both outcomes. To facilitate the ease of reviewing some of our favorite preventative measures, we have included them in a bulleted list below. Furthermore, **Figure 3** addresses one of the more pernicious problems with synchronous online live coding: how do you share your screen so students can follow along on a single, small computer display.

Given not everyone uses live coding in their courses, we compiled an additional list of tips and tricks to keep in mind when teaching in this fashion:

- **Pace:** Go slow! Be sure you give students time to catch up.
- **Explain:** Especially early on, explain every single thing you do. This includes how to run lines of code, saving scripts, etc.
- **Illuminate:** Don't correct errors you make without explaining them. New coders struggle troubleshooting and this is a great learning opportunity.
- **Encourage:** Normalize getting errors. It can even be good to intentionally make some common errors and then talk about them. Emphasize that getting and solving errors is part of the process!
- **Pause:** If students are following along on their own computers, pause frequently and ask students if they've had any "fun" errors. Thank them for sharing them and explain how you make corrections or find solutions when you have similar errors.

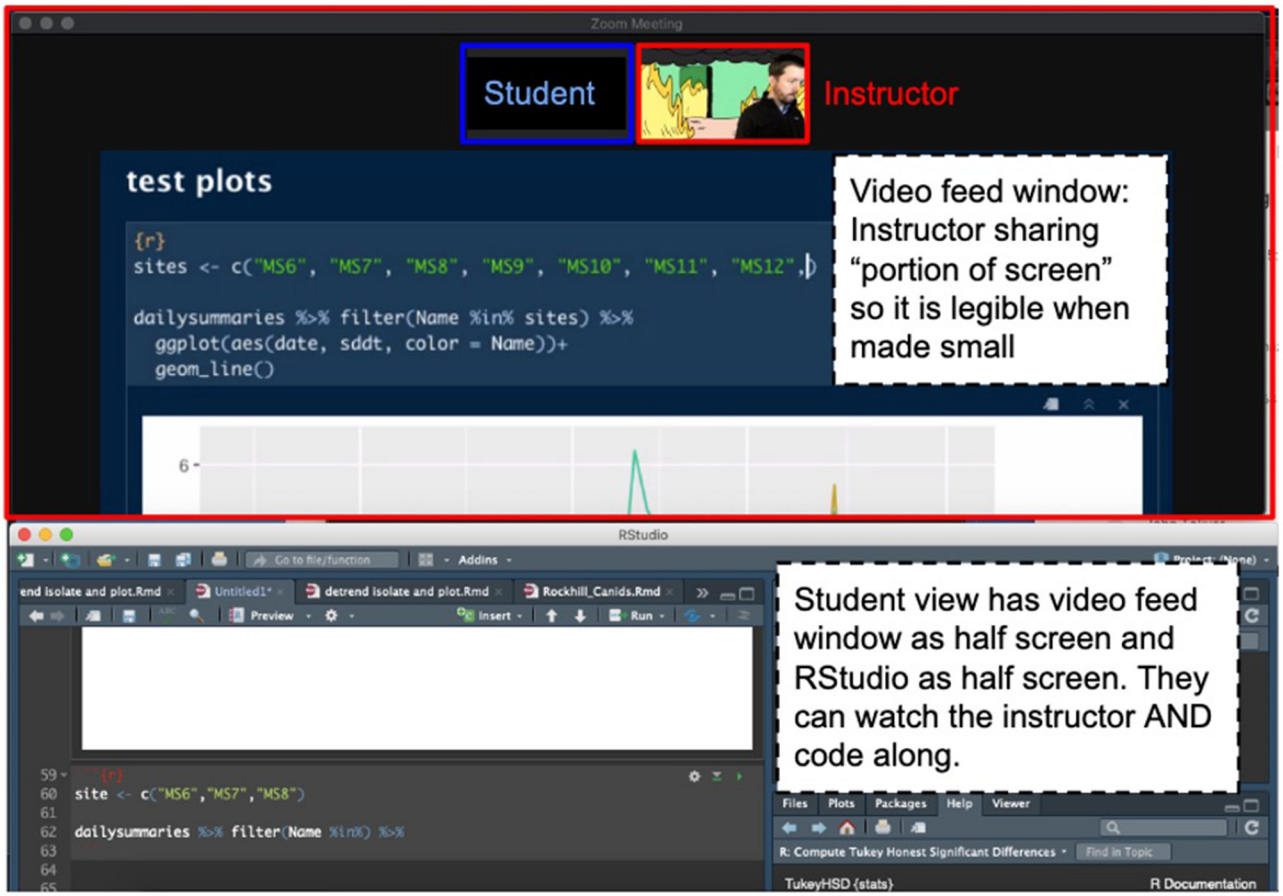

FIGURE 3 | In a virtual environment, encourage students to be mindful of how they set up their screen. In this figure, we show one setup that has worked well for the authors. The setup above shows an example of a student's single screen. In it, the student has their zoom window sized to fit the top half of the screen and RStudio sized to the bottom half. With this setup, the student can watch and code at the same time. It is critical for this to work that the instructor only shares part of their screen, allowing them to show just their code, making it legible even when a small window on an already small screen. We recommend sharing an image such as this at the start of the semester, especially for students participating in a virtual or hybrid learning environment.

- **Reflect:** Be careful with your language. Avoid saying things like "we simply do this", "we just do this", "this is easy" or "this is straightforward". Students hear this and if they are struggling, they may think they are not able to understand "easy" material and therefore get discouraged.
- **Avoid:** Don't make fun of spreadsheet programs (e.g., Excel). Students may have had trouble with them in the past. Many students will view learning to code as much harder than learning Excel and may therefore start to think they cannot be successful at learning to code.
- **Share:** When you have students complete a "challenge" or activity on their own, ask them to explain what they did to a partner before going over it in class. Then ask for a volunteer to explain their solution to the class. Explaining your code to someone else is a powerful learning tool.

BMP 5: Teach Students How to Help Themselves and Learn From Their Errors

One of the components of teaching coding that the authors of this piece are always seeking to improve is how to assist students in learning how to help themselves when their code doesn't work (e.g., receiving errors) or perform as expected. We have all had the experience when teaching a course—often on the first day of demoing coding and asking students to follow along—that a student says "my code isn't working". Likewise, we have had the same experience via email as students work through their first coding assignment. These four words are bound to be repeated to you, as they were to us, again and again.

In this context, there are several approaches that we use to teach students how to help themselves and learn from their errors. Above all, it is important to create a classroom environment where students feel comfortable asking for assistance and trust their instructors (Wang et al., 2021), and where encountering errors is normalized (see Section BMP 4: Do Live Coding). As instructors, we have shared with our students our own struggles learning to code, and how we have overcome these challenges; we always seek to be honest in how we represent our own experiences, to remind students that learning to code (and learning in general) is a process.

In the classroom, one of the best things we've found that we can do as instructors is to create a classroom environment where students feel comfortable asking questions and identifying that they are unsure of next steps (Sidelinger and Booth-Butterfield, 2010). One important consideration in this process is to normalize encountering errors (when you are teaching and hopefully live coding, see BMP 4). As experienced by the authors, you probably won't have to try hard to encounter errors that become teachable moments! In these moments, students have a chance to watch how you approach the debugging process, often via an internet search, another advantage of live coding, as described in Section BMP 4: Do Live Coding (Raj et al., 2018; Selvaraj et al., 2021). As an instructor, "getting stuck" is something to be upfront about ("it is going to happen to everyone!") and to discuss regularly as a class. One way to communicate this to students is to emphasize to them that learning to code is like learning to speak a different language. However, the goal is not memorization. Therefore, the overall objective of many discipline-specific courses that incorporate coding is not to teach students to code, but instead to teach them to problem solve in a coding environment. As educators who use coding in our research, one point we often explain to our students is that we, the instructors, rarely sit down and code from memory; instead, we discuss how we approach coding for our own projects—using example code (either ours or example code we find via internet search), and debugging (the process of finding and correcting errors in our code) via internet search.

One approach that many of us tried when first teaching coding was to attempt to solve each students' errors during in-class exercises. This amounted to a lot of stress on our parts—either moving from breakout room to breakout room or running around the classroom. Above all, we have learned to resist the urge to take the students' computer (or virtually, take over their computer via remote access) and fix the error. Instead, we encourage the student (or team) to explain what they are trying to do, and ask them questions to help they realize what they've done wrong.

For in-class exercises when only a few students have errors, one approach we have used in a virtual environment is to ask individual students to share their screens, and work as a class to spot the errors. This supports peer-to-peer interactions and reminds students that their peers can help them find their errors (instead of having them always come to the instructor). However, depending on the size of a given class and the length of the class period, this approach, and addressing all students who encounter errors, is often ineffective. An alternative, with benefits for all students, is to facilitate peer teaching, where students work in small groups of two or three and code and troubleshoot together (virtually, this can be accomplished via breakout rooms). This approach is known as pair programming and is widely lauded in the educational programming literature (McDowell et al., 2003; Hanks et al., 2011; Salleh et al., 2011). In pair programming, (often) two students work together to write code at a single workstation. Pair programming is a useful approach for both in-class activities as well as out-of-class assignments. The educational literature has shown numerous positive outcomes associated with this approach.

For in-class exercises, another effective practice is to have a signal that students can use to let you, the instructor, know when they are stuck. When in person, the Data Carpentries instructor training recommends the use of green and red stickies: a green sticker at the top of a computer monitor indicates no issues, while a red sticker indicates a problem has been encountered (The Carpentries, 2022). In the virtual classroom, the chat feature and emojis can be used in a similar way. Students may try to use the chat feature to message only the instructor; instead, encourage them to message everyone, and again, normalize everyone solving each other's errors and helping each other to learn together.

Outside the classroom, we have found it effective to dedicate portions of in-class time to discuss and develop with each class a procedure for how each student should go about getting help if they are stuck. This discussion serves two purposes: it indicates to students how they can best share information via email with their instructors if they are seeking help outside of office hours, and it can address how students can use educational resources to fix their own coding errors. As an instructor, it's worth considering how you prefer to assist students—is having them email their code preferred? Can you spot issues from code copy and pasted into an email? Be prescriptive about how students should go about asking questions, what expectations you have of approaches they should try before they reach out to you, and how they should explain their issue and their approach when they ask for help. It's also worth considering what level of help you're willing to give—a hint, or more.

As part of classroom instruction, the instructor should introduce students to the concept of debugging. The literature on "debugging"—a word that broadly describes identifying and fixing errors in code—is rich (McCauley et al., 2008). As emphasized by the literature, debugging must be taught—it is not a skill students will learn through the process of writing code alone (Kessler and Anderson, 1986; Carver and Risinger, 1987; Chmiel and Loui, 2004; McCauley et al., 2008). In addition to providing examples during class, instructors should introduce (and continue to remind students of) resources such as StackOverflow and DaniWeb, where students may be able to find discussions of those who have encountered (and solved) similar errors or to post their own questions. In this vein, we also recommend an assignment where students post their code and an error they are having to a website, to engage students in the process of intelligently framing a coding question for an online forum. It's worth pointing out to students that learning to code is equally important as learning how to problem solve their coding errors.

BMP 6: Know What Resources Are Available to You

When each of us sought to either build our courses or to add coding into our existing courses, one of the first things we did was to begin looking for existing resources on educational websites and as shared by colleagues on social media platforms. For this reason, we remind all readers who are approaching teaching a coding course (or incorporating coding into an existing course) that there are always resources available to support your needs.

TABLE 1 | Freely available resources that can be used to incorporate coding into hydrology courses.

Resource category	Title	Website
Introductory Resources	Swirl	https://swirlstats.com
	Basic Basics	https://rladiessydney.org/courses/ryouwithme/01-basicbasics-0/
	R for Data Science	https://r4ds.had.co.nz/
	CyberHelp at SESYNC	https://cyberhelp.sesync.org/lesson/
	Fundamentals of Data Visualization	https://clauswilke.com/dataviz/
	R for Cats	https://rforcats.net/
R and Hydrology	HydroInformatics	https://vt-hydroinformatics.github.io/
	Geocomputation with R	https://geocompr.robinlovelace.net/
	R for Water Resources Data Science	https://www.r4wrds.com/intro/index.html
	Hydrological Data and Modeling Resources in R	https://cran.r-project.org/web/views/Hydrology.html
Course Materials and Repositories	HydroShare	https://www.hydroshare.org/
	Earth Data Science by EarthLab	https://www.earthdatascience.org/
	Science Education Research Center (SERC)	https://serc.carleton.edu/index.html
	Data Carpentries Semester Course in Biology	https://datacarpentry.org/semester-biology/

Resources are all based around R.

We advocate perusing such available resources before developing your own assignments. In **Table 1**, we highlight several resources related to hydrologic sciences and popular coding languages; this is far from an exhaustive list but represents the tools we are aware of and have often used in the classroom. We recommend investigating materials that introduce the R basics (or the basics of any given language) and that are interactive (e.g., swirl), as they are great for initial homework assignments or supplementing instruction. Depending on the discipline you are teaching in, hydrology or otherwise, there are likely to be other repositories for course assignments and modules (**Table 1**). There are also many educators who host such material on their personal websites, though these materials may be harder to find. Finally, we encourage anyone pursuing this route to also network amongst your colleagues, as many are more than willing to share their course materials, and to eventually be willing to share your own materials.

Though resources can limit the time you spend preparing educational materials, the process of learning to effectively *teach* coding will take time. In addition to these resources, the authors also wish to highlight and recommend training offered through Data Carpentries (see: https://carpentries.github.io/instructor-training/). This type of training is not focused on how to teach any specific computing language but can be thought of as a training in how to effectively teach coding.

BMP 7: Align Hydrologic Content With Coding Principles

This is the part of teaching we as instructors all struggled with the most, as educators teaching course content infused with coding. When students start such a course, they often haven't used R before, and, for many, are taking their first course in hydrology, but are expected to possess working knowledge of both by the end of the semester. If planned well, instructors can introduce the skills needed to perform the analyses they are teaching. As instructors, we are still perfecting our approaches to this, and have found that reflecting and taking good notes after each lesson has helped us iterate our approaches and our courses (especially in terms of where students either immediately grasped a concept, or a place where they collectively struggled). Based on our own experiences, **Table 2** shows examples from our courses that align hydrologic course material and R coding.

BMP 8: Assess Student Learning Often and With Low Stakes Interactions

In our coding courses, and in agreement with the educational literature, we have found providing low stakes assessments (anecdotally) appears to improve the classroom experience for students as well as overall learning outcomes. Low stakes assessments are those that provide students with an opportunity to test their knowledge and receive feedback but via an assignment or quiz that constitutes only a small percentage of each student's overall grade. We all have used low stakes assessments that encourage students to actively engage with the material by providing opportunities for repetition needed to build skill competencies in a structured environment. Moreover, such assessments provide instructors with real-time feedback on the status of student learning (i.e., formative assessments). Below are several assessments that we found useful in our courses that draw from the educational literature.

TABLE 2 | Matching hydrologic concepts with coding concepts in a hydrology course.

Hydrologic concepts	Coding concepts	Useful R libraries
Differences in hydrographs across climate regions	Commands and options for creating a figure	*ggplot2*
Flow duration curves	Creating vectors; Basic data wrangling (e.g., sorting)	*dplyr*
Computing runoff ratios	Basic data wrangling (e.g., aggregating and grouping)	*dplyr*
Stage-discharge relationships	Generating statistical models	*stats*
Linear reservoir modeling	*for* loops, *if/then/else* statements	-
Estimating potential vs. actual evapotranspiration	writing functions	-
Watershed delineation	Geospatial analysis	*Whitebox, tmap*
Extracting watershed precipitation	Geospatial analysis	*prism*
Hydrologic model calibration and optimization	Developing workflows	*TUWmodel, rtop*

Low Stakes and No Stakes Assessments

In our coding courses, we have found short quizzes at the beginning and end of class (∼5 questions over 5 min) to be an effective low stakes assessment (Narloch et al., 2006; Lyle and Crawford, 2011). These assessments can be a mix of hydrologic and coding-based questions, and we suggest the quizzes be very similar in format (if not identical) from week to week. This provides students opportunity for additional repetition and to show improvement. Optional questions could include writing a short snippet of code, answering multiple choice questions, or drawing a conceptual model describing hydrologic process.

Retrieval Practice

In conjunction with pre- or post-class quizzes discussed above, we suggest incorporating retrieval practice throughout class. For example, one approach used by one of the authors is to provide a code chunk to the class and instruct students to look for errors, or to engage students in writing pseudo-code (i.e., provide a written summary of what the code does; also described in the computer science literature as verbal algorithm specification) as part of their assignments. Importantly, these activities can be done as a class, in small groups, or individually—providing flexibility under circumstances when the classroom may transition between virtual, in-person, or hybrid modes.

Two-Stage Exams

As introduced by Zipp (2007), two-stage exams emphasize cooperative learning for testing, turning exams into not only an assessment, but an opportunity for learning. In coding courses, we recommend the use of two-stage exams. In this approach, students complete their exam, and the exam is returned to them by the next time the class meets. During that next class meeting, students are given time in class to work in teams to correct everything that is wrong. This allows students who understood the material to learn it better by teaching it to others and allows those students who didn't perform as well to re-engage with the material and learn from their peers. There are many examples in the educational literature documenting the successes of two-stage exams and providing recommendations for how to incorporate this practice into various types of classrooms (Knierim et al., 2015; Bruno et al., 2017).

BMP 9: Learn Evidence-Based Effective Teaching Best Practices

While none of us consider ourselves experts when it comes to evidence-based teaching practices, we have all found immense value in engaging with this literature to improve our awareness of these techniques and to test these techniques in our classrooms. There are numerous well researched and well-developed strategies for effective teaching. Though the literature on the topic continues to expand our understanding of how individuals learn, there are also many commonly accepted best practices. From our experience, one of the best ways to learn about these best practices, both what they are and how they work, is to read books that summarize them. Diving into the literature can be overwhelming and difficult to translate into classroom actions. Instead, we recommend instructors find a book. *Small Teaching* (Lang), *Teaching at Its Best* (Nilson), *The Chicago Guide to College Science Teaching* (McGlynn), and many others offer neatly summarized best practices, examples, and explanations based on the literature. Likewise, instructors should familiarize themselves with Universal Design for Learning (UDL). UDL features an evolving set of guidelines that are aimed at helping instructors meet student needs (Courey et al., 2013; CAST, 2018). Many of the recommendations contained in this article are inspired by UDL principles in the context of coding education. UDL is ever-evolving, thus revisiting UDL resources year after year is likely to be a good use of time.

Another resource available to build awareness of effective teaching best practices are teaching workshops offered by college and university teaching centers or professional organizations. In addition to being engaging ways to learn more about teaching and get new ideas for your classroom, they can be a great way to meet others at your institution or in professional organizations who share your interest in the subject. From our experiences, we found that engaging with multiple books and workshops was the best way to identify best practices for our classrooms. Additionally, it helps to update or annotate your course notes, class schedule, or other material immediately after you engage with these new materials, lest they be lost to the thousand other demands on your time.

Though we include a discussion of student self-efficacy earlier in this piece, we note in this section that educator self-efficacy is equally important to supporting student self-efficacy. Thus,

an awareness of and practice with evidence-based teaching strategies will not only improve an educators' experience in the classroom, but will also support their students' self-efficacy, motivation, and learning (Woolfolk Hoy, 2004). In this frame, we encourage educators teaching coding to experiment with educational activities in the classroom: try a new approach, or strategy, and reflect on what went well (or what didn't go well). While not every strategy that we have used in classrooms has always worked, or achieved the intended outcome, such experiments are useful information for improving approaches through time.

BMP 10: Grow Your Knowledge Every Year

Best practices for teaching are ever evolving. In this vein, we have all found it incredibly useful to engage in training on teaching each year, whether that is reading a book, attending a seminar, or reading a few articles in the peer-reviewed literature. We feel it helps keep us current and gives us new ideas to bring into the classroom, which improves our teaching and helps keep teaching interesting and exciting for us. As core practices in science education are constantly evolving, this also helps us to keep practicing and learning new skills.

CONCLUSIONS

Teaching is challenging. However, it is not a magical mystery show. Just as we don't expect our students to be automatically good at coding, we cannot expect to be automatically good at teaching. Teaching is a skill that can be improved through practice and training. As we highlight in this article, there are many resources available to aid instructors on our journey, including the recommendations presented here. These recommendations are aimed to help any instructor consider how to approach teaching students to code, whether under virtual environments or otherwise. We caution that this is not an exhaustive list and represents a set of core practices that we expect to evolve over time, both within our classrooms and within the hydrology community.

Providing instruction during the pandemic has been (and continues to be) challenging for so many reasons. However, this experience was crucial for us to identify how we could improve our teaching around coding, given all the changes and circumstances that virtual teaching and learning presented. Above all, we believe this experience, as reflected by these recommendations, will lead to more effective and inclusive teaching in the future.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

CK, JG, and CJ contributed to the conception of the manuscript. All authors contributed to various sections of the manuscript, read, and approved the submitted version.

FUNDING

Funding was provided by Lafayette College and a Syracuse University CUSE Grant to CK.

ACKNOWLEDGMENTS

The authors wish to thank the students in their courses who made this article possible and who helped form our experiences teaching coding in the classroom. We also appreciate the thoughtful feedback we received from the three reviewers listed on this article.

REFERENCES

Anghileri, J. (2006). Scaffolding practices that enhance mathematics learning. *J. Math. Teach. Educ.* 9, 33–52. doi: 10.1007/s10857-006-9005-9

Artino, A. R. (2008). Motivational beliefs and perceptions of instructional quality: predicting satisfaction with online training. *J. Comput. Assist. Learn.* 24, 260–270. doi: 10.1111/j.1365-2729.2007.00258.x

Ballen, C. J., Wieman, C., Salehi, S., Searle, J. B., and Zamudio, K. R. (2017). Enhancing diversity in undergraduate science: self-efficacy drives performance gains with active learning. *CBE Life Sci. Educ.* 16, ar56. doi: 10.1187/cbe.16-12-0344

Bandura, A. (1977). Self-efficacy: toward a unifying theory of behavioral change. *Psychol. Rev.* 84, 191–215. doi: 10.1037/0033-295X.84.2.191

Brown, N. C. C., and Wilson, G. (2018). Ten quick tips for teaching programming. *PLOS Comput. Biol.* 14, e1006023. doi: 10.1371/journal.pcbi.1006023

Bruno, B. C., Engels, J., Ito, G., Gillis-Davis, J., Dulai, H., Carter, G., et al. (2017). Two-stage exams: a powerful tool for reducing the achievement gap in undergraduate oceanography and geology classes. *Oceanography* 30, 198–208. doi: 10.5670/oceanog.2017.241

Carnevale, A. P., Smith, N., and Melton, M. (2011). *Science, Technology, Engineering, and Mathematics*. Georgetown University Center on Education and the Workforce. Available online at: https://cew.georgetown.edu/cew-reports/stem/#resources (accessed January 7, 2022).

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. *ACM SIGCSE Bull.* 38, 27–31. doi: 10.1145/1124706.1121352

Carver, S., and Risinger, S. (1987). "Improving children's debugging skills," in *Empirical Studies of Programmers: Second Workshop*, eds G. Olson, S. Sheppard, and E. Soloway (Norwood, NJ: Ablex) 147–171.

CAST (2018). *Universal Design for Learning Guidelines version 2.2*. Available online at: https://udlguidelines.cast.org/ (accessed January 31, 2022).

Chmiel, R., and Loui, M. C. (2004). Debugging: from novice to expert. *ACM SIGCSE Bull.* 36, 17–21. doi: 10.1145/1028174.971310

Corradini, I., Lodi, M., and Nardelli, E. (2018). "An investigation of Italian primary school teachers' view on coding and programming," in *11th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2018* (St. Petersburg), 228–243. doi: 10.1007/978-3-030-02750-6_18.hal-01913059

Courey, S. J., Tappe, P., Siker, J., and LePage, P. (2013). Improved lesson planning with universal design for learning (UDL). *Teach. Educ. Spec. Educ.* 36, 7–27. doi: 10.1177/0888406412446178

De Cicco, L. A., Hirsch, R. M., Lorenz, D., Watkins, W. D., and Johnson, M. (2021). *dataRetrieval: R Packages for Discovering and Retrieving Water Data Available from Federal Hydrologic Web Services*. Reston, VA: U.S. Geological Survey. Available online at: https://code.usgs.gov/water/dataRetrieval (accessed January 17, 2022).

Dirzyte, A., Sederevičiute-Pačiauskiene, Ž., Šliogeriene, J., Vijaikis, A., and Perminas, A., Kaminskis, L., et al. (2021). Peer-to-peer confirmation, positive automatic thoughts, and flourishing of computer programming e-learners. *Sustainability* 13, 11832. doi: 10.3390/su132111832

Fencl, H. S., and Scheel, K. R. (2005). Research and teaching: engaging students - an examination of the effects of teaching strategies on self-efficacy and course climate in a nonmajors physics course. *J. Coll. Sci. Teach.* 35, 20–24

Finley, A. (2021). *How College Contributes to Workforce Success: Employer Views on What Matters Most*. Washington, DC: Association of American Colleges and Universities. Available online at: https://www.aacu.org/research/how-college-contributes-to-workforce-success (accessed January 18, 2022).

Govender, I., Govender, D. W., Havemga, M., Mentz, E., Breed, B., Dignum, F., et al. (2014). Increasing self-efficacy in learning to program : exploring the benefits of explicit instruction for problem solving. *J. Transdiscipl. Res. South Afr.* 10, 187–200. doi: 10.10520/EJC154539

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., and Zander, C. (2011). Pair programming in education: a literature review. *Comput. Sci. Educ.* 21, 135–173. doi: 10.1080/08993408.2011.579808

Harland, T. (2003). Vygotsky's zone of proximal development and problem-based learning: linking a theoretical concept with practice through action research. *Teach. High. Educ.* 8, 263–272. doi: 10.1080/1356251032000052483

Honicke, T., and Broadbent, J. (2016). The influence of academic self-efficacy on academic performance: a systematic review. *Educ. Res. Rev.* 17, 63–84. doi: 10.1016/j.edurev.2015.11.002

Jacobs, C. T., Gorman, G. J., Rees, H. E., and Craig, L. E. (2016). Experiences with efficient methodologies for teaching computer programming to geoscientists. *J. Geosci. Educ.* 64, 183–198. doi: 10.5408/15-101.1

Kessler, C., and Anderson, J. (1986). "A model of novice debugging in LISP," in *Empirical Studies of Programmers*, eds E. Soloway and S. Iyengar, (Norwood, NJ: Ablex) 198–212.

Killpack, T. L., and Melón, L. C. (2016). Toward inclusive STEM classrooms: what personal role do faculty play? *CBE Life Sci. Educ.* 15, es3. doi: 10.1187/cbe.16-01-0020

Kirschner, P. A., and Hendrick, C. (2020). *How Learning Happens: Seminal Works in Educational Psychology and What They Mean in Practice*. London: Routledge. doi: 10.4324/9780429061523

Knierim, K., Turner, H., and Davis, R. K. (2015). Two-stage exams improve student learning in an introductory geology course: logistics, attendance, and grades. *J. Geosci. Educ.* 63, 157–164. doi: 10.5408/14-051.1

Lewis, F., Edmonds, J., and Fogg-Rogers, L. (2021). Engineering science education: the impact of a paired peer approach on subject knowledge confidence and self-efficacy levels of student teachers. *Int. J. Sci. Educ.* 43, 793–822. doi: 10.1080/09500693.2021.1887544

Lin, Y.-T., Wang, M.-T., and Wu, C.-C. (2019). Design and implementation of interdisciplinary STEM instruction: teaching programming by computational physics. *Asia Pac. Educ. Res.* 28, 77–91. doi: 10.1007/s40299-018-0415-0

Loo, C. W., and Choy, J. L. F. (2017). Sources of self-efficacy influencing academic performance of engineering students. *Am. J. Educ. Res.* 1, 86–92. doi: 10.12691/education-1-3-4

Lyle, K. B., and Crawford, N. A. (2011). Retrieving essential material at the end of lectures improves performance on statistics exams. *Teach. Psychol.* 38, 94–97. doi: 10.1177/0098628311401587

Lyon, J. A., and Magana, A. J. (2020). Computational thinking in higher education: a review of the literature. *Comput. Appl. Eng. Educ.* 28, 1174–1189. doi: 10.1002/cae.22295

McBride, E., Oswald, W. W., Beck, L. A., and Vashlishan Murray, A. (2020). "I'm just not that great at science": Science self-efficacy in arts and communication students. *J. Res. Sci. Teach.* 57, 597–622. doi: 10.1002/tea.21603

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., et al. (2008). Debugging: a review of the literature from an educational perspective. *Comput. Sci. Educ.* 18, 67–92. doi: 10.1080/08993400802114581

McDowell, C., Hanks, B., and Werner, L. (2003). "Experimenting with pair programming in the classroom," in *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education ITiCSE'03* (New York, NY: Association for Computing Machinery), 60–64. doi: 10.1145/961511.961531

McGlynn, T. (2020). *The Chicago Guide to College Science Teaching*. Chicago, IL: University of Chicago Press. doi: 10.7208/9780226542539

Medeiros, R. P., Ramalho, G. L., and Falcão, T. P. (2019). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Trans. Educ.* 62, 77–90. doi: 10.1109/TE.2018.2864133

Meral, M., Colak, E., and Zereyak, E. (2012). The relationship between self-efficacy and academic performance. *Proc. Soc. Behav. Sci.* 46, 1143–1146. doi: 10.1016/j.sbspro.2012.05.264

Narloch, R., Garbin, C. P., and Turnage, K. D. (2006). Benefits of prelecture quizzes. *Teach. Psychol.* 33, 109–112. doi: 10.1207/s15328023top3302_6

National Academies of Sciences, Engineering, and Medicine (2016). *Developing a National STEM Workforce Strategy: A Workshop Summary*. National Academies Press.

National Academies of Sciences, Engineering, and Medicine (2018). *How People Learn II: Learners, Contexts, and Cultures*. National Academies Press.

National Research Council (2000). *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. National Academies Press.

Nesen, Y., Fowler, B., and Vegas, E. (2021). *How Italy Implemented Its Computer Science Education Program*. Center for Universal Education at Brookings. Available online at: https://www.brookings.edu/wp-content/uploads/2021/10/How-Italy-implemented-its-CS-education-program_FINAL.pdf (accessed April 8, 2022).

Pajares, F. (1996). Self-efficacy beliefs and mathematical problem-solving of gifted students. *Contemp. Educ. Psychol.* 21, 325–344. doi: 10.1006/ceps.1996.0025

Raj, A. G. S., Gu, P., Zhang, E., Annie R, A. X., Williams, J., Halverson, R., et al. (2020). "Live-coding vs static code examples: which is better with respect to student learning and cognitive load?" in *Proceedings of the Twenty-Second Australasian Computing Education Conference ACE'20* (New York, NY: Association for Computing Machinery), 152–159. doi: 10.1145/3373165.3373182

Raj, A. G. S., Patel, J. M., Halverson, R., and Halverson, E. R. (2018). "Role of live-coding in learning introductory programming," in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research Koli Calling'18* (New York, NY: Association for Computing Machinery), 1–8. doi: 10.1145/3279720.3279725

Rubin, M. J. (2013). "The effectiveness of live-coding to teach introductory programming," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education SIGCSE'13* (New York, NY: Association for Computing Machinery), 651–656. doi: 10.1145/2445196.2445388

Ryan, R. M., and Deci, E. L. (2020). Intrinsic and extrinsic motivation from a self-determination theory perspective: definitions, theory, practices, and future directions. *Contemp. Educ. Psychol.* 61, 101860. doi: 10.1016/j.cedpsych.2020.101860

Salleh, N., Mendes, E., and Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: a systematic literature review. *IEEE Trans. Softw. Eng.* 37, 509–525. doi: 10.1109/TSE.2010.59

Samiullah, M. (1995). Effect of in-class student–student interaction on the learning of physics in a college physics course. *Am. J. Phys.* 63, 944–950. doi: 10.1119/1.18038

Schunk, D. H., and Pajares, F. (2002). "The development of academic self-efficacy," in *Development of Achievement Motivation* (San Diego, CA: Academic Press), 15–31. doi: 10.1016/B978-012750053-9/50003-6

Selvaraj, A., Zhang, E., Porter, L., and Soosai Raj, A. G. (2021). "Live coding: a review of the literature," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY: Association for Computing Machinery), 164–170.

Shannon, A., and Summet, V. (2015). Live coding in introductory computer science courses. *J. Comput. Sci. Coll.* 31, 158–164. Available online at: https://dl.acm.org/doi/10.5555/2831432.2831457 (accessed January 16, 2022).

Sharma, P., and Hannafin, M. J. (2007). Scaffolding in technology-enhanced learning environments. *Interact. Learn. Environ.* 15, 27–46. doi: 10.1080/10494820600996972

Sidelinger, R. J., and Booth-Butterfield, M. (2010). Co-constructing student involvement: an examination of teacher confirmation and student-to-student

connectedness in the college classroom. *Commun. Educ.* 59, 165–184. doi: 10.1080/03634520903390867

Sollitto, M., Johnson, Z. D., and Myers, S. A. (2013). Students' perceptions of college classroom connectedness, assimilation, and peer relationships. *Commun. Educ.* 62, 318–331. doi: 10.1080/03634523.2013.788726

Steele, C. M., Spencer, S. J., and Aronson, J. (2002). "Contending with group image: the psychology of stereotype and social identity threat," in *Advances in Experimental Social Psychology*, ed. M. P. Zanna (Waltham, MA: Academic Press), 379–440. doi: 10.1016/S0065-2601(02)80009-0

Stoeckel, M. R., and Roehrig, G. H. (2021). Gender differences in classroom experiences impacting self-efficacy in an AP Physics 1 classroom. *Phys. Rev. Phys. Educ. Res.* 17, 020102. doi: 10.1103/PhysRevPhysEducRes.17.020102

The Carpentries (2022). *Instructor Training*. Available online at: https://carpentries.github.io/instructor-training/ (accessed February 1, 2022).

The Royal Society (2017). *After the Reboot: Computing Education in UK Schools*. London. Available online at: https://royalsociety.org/-/media/policy/projects/computing-education/computing-education-report.pdf (accessed April 8, 2022).

Valente, J. A., and de Almeida, M. E. B. (2020). Políticas de tecnologia na educação no Brasil: Visão histórica e lições aprendidas. *Educ. Policy Anal. Arch.* 28, 94. doi: 10.14507/epaa.28.4295

Van Merrienboer, J. J. G., and Krammer, H. P. M. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instr. Sci.* 16, 251–285. doi: 10.1007/BF00120253

Wang, C., Cavanagh, A. J., Bauer, M., Reeves, P. M., Gill, J. C., Chen, X., et al. (2021). A framework of college student buy-in to evidence-based teaching practices in STEM: the roles of trust and growth mindset. *CBE Life Sci. Educ.* 20, ar54. doi: 10.1187/cbe.20-08-0185

Wing, J. M. (2006). Computational thinking. *Commun. ACM* 49, 33–35. doi: 10.1145/1118178.1118215

Woolfolk Hoy, A. (2004). *Self-Efficacy in College Teaching*. Available online at: https://cft.vanderbilt.edu/wp-content/uploads/sites/59/vol15no7_self_efficacy.htm (accessed April 3, 2022).

Yan, Y. (2017). Teaching programming skills to finance students: how to design and teach a great course. *Financ. Innov.* 3, 32. doi: 10.1186/s40854-017-0081-x

Zipp, J. F. (2007). Learning by exams: the impact of two-stage cooperative tests. *Teach. Sociol.* 35, 62–76. doi: 10.1177/0092055X0703500105

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.