Ten Quick Tips For Teaching Using Participatory Live Coding

Lex Nederbragt, Greg Wilson

March 2020, version 0.3

Introduction

What is participatory live coding?

Participatory live coding is a technique where a teacher or instructor does live programming while the learners simultaneously copy and execute the exact code or commands that are being written. The instructor reads what is being typed out loud, explaining the different elements and principles. Instructor and learners all execute the commands or program, leading to an immediate evaluation of the results. Learners thus 'code-along' with the instructor. There are frequent, often short, exercises, where learners are asked to solve a small relevant problem on their own.

This approach aims to be an improvement on teaching programming through lecturing showing static code, or relying on learners reading a textbook or compendium. What is taught is immediately applied rather than just shown on a slide or on paper. It also slows the instructor down, giving learners more time to actively engage with the material before moving on to the next concept. Importantly, the thought process behind coding can also be made explicit. Learner's questions can immediately be answered and misconceptions corrected by coding them. Exercises enable immediate practice using the material. The participatory, 'code-along' aspect is important to help novices become active practitioners, rather than passive observers of the programming process.

Crucially, the technique also allows for teaching handling of mistakes. Beyond deliberately introducing mistakes during the live coding, instructors will often make unplanned mistakes. Novice learners are likely to make many such mistakes themselves, and diagnosing and solving mistakes is an integral aspect of learning programming.

Participatory live coding for teaching programming should not be confused with Live Coding used to demonstrate software (for example, at a conference, with an audience passively observing), Live Streaming programming [1], or used as a form

of performing art (e.g. while creating computer music [2]). A video recording demonstrating the technique can be found here: https://vimeo.com/139316669.

Who uses it

Participatory live coding is the main teaching method in workshops organised by the global non-profit called The Carpentries,[1] the umbrella organisation for Software Carpentry, Data Carpentry and Library Carpentry [3]. Increasingly, university courses involving the teaching of programming or related techniques employ the method [4].[2]

There is a limited body of research on the effectiveness of Live coding in programming education. Most studies focus on non-participatory live coding, demonstrating the programming process and contrasting this with the use of static code on slides (see [5] and references therein, as well as [6]).

So far, the technique show that live-coding is as good as if not better than using static code examples [5,7] and thus is a recommended approach for teaching programming [8,9].

The ten quick tips described below are aimed at those interested in applying the technique in to their own teaching. They are meant to complement the "Ten quick tips for delivering programming lessons" [8].

Tip 1. Go slowly.

Say out loud what you are doing while you do it for every command you type, every word of code you write, and every menu item or website button you click. Then, point to the command and its output on the screen and go through it a second time. This allows learners to catch up and check their understanding: if they are watching you type and trying to type the same code in themselves, they don't have time to think about what they're doing. As you go through it a second time, they can check their understanding and correct any small typos they may have made. This is particularly important for learners who may not be fluent in the language of instruction, or who may have hearing, vision, or mobility impairments.

If the output of your command or code makes what you just typed disappear from view, scroll back up so learners can see it again.

Do not copy-paste code from your lesson material and/or ask your learners to do the same. It's all too easy to go too fast without explaining the thought process behind the code.

[1] https://carpentries.org

[2] https://lexnederbragt.com/blog/2017-12-17-experiences-with-the-first-edition-of-introduction-to-computational-modelling-for-the-biosciences/

Tip 2. Mirror your learner's environment.

Try to create an environment that is as similar as possible to what your learners have. If learners have to work in a different environment that you adds a mental effort that does not contribute to learning. Cognitive psychological theory calls this extraneous cognitive load [10]. You may have personalised your environment with a very simple or rather fancy Unix prompt, colour schemes for your development environment, keyboard shortcuts, etc.

Your learners usually won't have all of this, so try to create an environment that mirrors what your learners have.

Similarly, avoid using keyboard shortcuts as these will hide the action(s) you are performing and learners may not know about them. If you must use them, use a keystroke visualizer tool or your computer's accessibility tools to echo keystrokes to the screen.

Some instructors create a separate 'bare-bone' user (login) account on their laptop or a separate 'teaching-only' account on the service being taught (e.g., Github). It could be a benefit if both instructor and learners use the exact same software (terminal or code development environment), however this may incur some effort to get the software installed on the computers that students use. Using a cloud-based solution may be an alternative to ensure all involved have the exact same setup.

Tip 3. Be seen and heard.

If you are physically able to stand up for a couple of hours, do it while you are teaching. When you sit down, you may appear hidden for those sitting in the back rows. Standing makes the experience more interactive and less monotonous, and draws the learners' attention away from their screens to you, which helps getting the point you are making across. Standing also encourages you to look at your audience rather than your screen: if you are sitting, you are likely to do what you normally do when you are sitting, which is look at your own computer. It helps to have a high table/standing desk or lectern so you can have your laptop at a comfortable height for typing.

Regardless of whether you are standing or sitting, make sure to move around as much as reasonable. For example, you could walk to the screen to point something out or draw something on the whiteboard (see below).

Even though you may have a good voice and know how to use it well, it may be an advantage to use a microphone, especially if the room is equipped with one. Your will tire your voice less, and you increase the chance of people with hearing difficulties being able to follow the teaching.

Tip 4. Use the screen(s) wisely.

Use a big font and maximize the window. Note with a large font, you may have fewer columns and rows than you're used to, and you should design examples with this in mind (or at least test them). A black font on a white background works better than a light font on a dark background. When the bottom of the projector screen is at the same height or below the heads of the learners, people in the back won't be able to see the lower parts, so resize the window(s) you use on your computer (drawing up the bottom) to compensate.

Pay attention to the lighting: no lights should directly shine on the presenter's screen.

It is even more important than when presenting slides that all learners can see the relevant portions of the screen as they may want to copy exactly what you have typed or see what you are pointing to.

If you can get a second screen, use it! It may require its own PC or laptop, so you may need to ask a helper to control it. You can use the second screen to show illustrations or the lesson material.

Tip 5. Avoid being disturbed.

Turn off notifications on your laptop and phone, such as those from social media, email, etc. Seeing notifications flash by on the screen distracts you as well as the learners, and may even result in awkward situations when a message pops up you'd rather not have others see.

Tip 6. Use illustrations - even better, draw them.

Lesson material often come with illustrations, and these may help learners to understand the stages of the lesson and to organize the material. What can work really well is when you as instructor generate the illustrations on the whiteboard as you progress through the material. This allows you to build up diagrams, making them increasingly complex in parallel with the material you are teaching. Presenting complementary information using visual and verbal representations helps learning (so-called "dual coding" [11]). Diagramming helps learners understand the material, makes for a more lively workshop (you'll have to move between your computer and the whiteboard), and gathers the learners' attention to you as well.

Tip 7. Stick to the lesson material.

When getting started with participatory live coding when teaching, it is advised to use a well-developed and tested lesson. Examples of such lessons specifically

written for teaching using this teachnique are the lessons from The Carpentries.[3] It may be tempting to deviate from your material because you would like to show a neat trick or demonstrate some alternative way of doing something, but there is always a fair chance you'll run into something unexpected that you then have to explain. It is thus advised not to improvise until you are familiar enough with the material and want to explain something with a low risk of failure. Use printouts of the lesson material during teaching, or alternatively use a second device (tablet or laptop) on which you can view your notes. Consider the use of a timer for exercises: they help keep yourself honest when you tell learners they have 5 minutes for an exercise. Sometimes a question or a "what ifs?" comes up that you'd like to address but need some time to sort through. Collect these, for example on sticky notes or ask learners to add them to a shared online document that they all can edit. This way, you can think about these while learners are doing exercises and answer them afterwards. You then don't disrupt the flow of the lesson while still showing that you take the learner's questions seriously.

Tip 8. Embrace your mistakes.

No matter how well prepared you are, you will make mistakes: typos are hard to avoid, you may overlook something from the lesson instructions, etc. This is not really a problem, and can actually be turned into a teachable moment: novices are going to spend at least some their time making similar mistakes, but how to deal with the is left out of most textbooks. Experiencing the instructor making a mistake allows learners to see how to diagnose and correct them, and gives the learners permission to make and share theirs. This way of dealing with mistakes is so-called "positive error framing" which has shown to be beneficial for learning [12].

Tip 9. Get real-time feedback.

Give each learner two sticky notes of different colours, e.g., blue and yellow, checking with your learners if all of them can distinguish between them (some people are blue-yellow colorblind). These can be held up for voting, but their real use is as status flags. If someone has completed an exercise, they put the yellow sticky note on their laptop; if they run into a problem and need help, they put up the blue one. This is better than having people raise their hands because they can keep working while their flag is raised, while signalling to any available helpers who to go to.

Also, the use of sticky notes allows the instructor to quickly see from the front of the room what state the class is in. Sometimes a blue sticky note involves a technical problem that takes a bit more time to solve. To prevent this issue slowing down the whole class too much, use the occasion to take the small break

[3] https://carpentries.org/workshops-curricula

you had planned to take a bit later, giving yourself or any helpers you may have time to fix the problem.

Tip 10. Turn learners into co-instructors.

During participatory live coding, learners are actively coding along with the instructor. You can engage them even more in different ways. For example, have learners call out the next line of code that they think you as instructor should type next. It helps you to understand any misunderstandings learners might have, as well as having them practice applying the material taught. You can also ask them to take notes collaboratively, using an online note-taking document that they all can edit. Having learners discuss and verbalize the material they just learned in their own words helps solidify their knowledge.

Conclusion

Participatory live coding is used successfully by thousands of instructors all over the world teaching programming, the use of the Unix shell or version control. It is increasingly being used in undergraduate teaching. It takes some practice to get used to presenting material this way. But after a few tries, most people feel it becomes natural, and rewarding as you interact with your learners in a wholly different way then when presenting slides.

Teaching is performance art and can be rather serious business. Don't let this scare you: it is OK to add an element of play, i.e., to use humor and improvisation to liven up the teaching. How much you are able and willing to do this is really a matter of personality and taste as well as experience. It becomes easier when you are more familiar with the material, allowing you to relax more. Choose your words and actions wisely, though. Remember that you want the learners to have a welcoming experience and a positive learning environment: a misplaced joke can ruin this in an instance. Start small: just saying "that was fun" after something worked well is a good start.

Sources for these ten quick tips and further reading

These tips were developed in the context of Software Carpentry and a first edition appeared on their blog.[4] They have become part of the Carpentries instructor training materials.[5] These also use example videos contrasting live coding done

[4] https://software-carpentry.org/blog/2016/04/tips-tricks-live-coding.html
[5] https://carpentries.github.io/instructor-training

poorly[6] and live coding done well.[7] [13] has a section on live coding.[8]

Acknowledgements

The authors wish to thanks Dr David Martin (University of Dundee, Scotland) for suggesting the word 'participatory' to distinguish this form of live coding from other forms.

References

1. Haaranen L. Programming as a Performance: Live-streaming and Its Implications for Computer Science Education. Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education. Bologna, Italy: Association for Computing Machinery; 2017. pp. 353–358. doi:10.1145/3059009.3059035

2. Collins N, McLEAN A, Rohrhuber J, Ward A. Live coding in laptop performance. Organised Sound. 2003;8: 321–330. doi:10.1017/S135577180300030X

3. Wilson G. Software Carpentry: Lessons learned. F1000Research. 2016;3: 62. doi:10.12688/f1000research.3-62.v2

4. Johnston L, Bonsma-Fisher M, Ostblom J, Hasan A, Santangelo J, Coome L, et al. A graduate student-led participatory live-coding quantitative methods course in R: Experiences on initiating, developing, and teaching. Journal of Open Source Education. 2019;2: 49. doi:10.21105/jose.00049

5. Raj AGS, Patel JM, Halverson R, Halverson ER. Role of Live-coding in Learning Introductory Programming. Proceedings of the 18th Koli Calling International Conference on Computing Education Research. Koli, Finland: Association for Computing Machinery; 2018. pp. 1–8. doi:10.1145/3279720.3279725

6. Raj AGS, Gu P, Zhang E, R AXA, Williams J, Halverson R, et al. Live-coding vs Static Code Examples: Which is better with respect to Student Learning and Cognitive Load? Proceedings of the Twenty-Second Australasian Computing Education Conference. Melbourne VIC Australia: ACM; 2020. pp. 152–159. doi:10.1145/3373165.3373182

7. Rubin MJ. The Effectiveness of Live-coding to Teach Introductory Programming. Proceeding of the 44th ACM Technical Symposium on Computer Science Education. Denver, Colorado, USA: ACM; 2013. pp. 651–656. doi:10.1145/2445196.2445388

[6] https://youtu.be/bXxBeNkKmJE
[7] https://youtu.be/SkPmwe_WjeY
[8] Also accessible at https://teachtogether.tech/#s:performance-live

8. Brown NCC, Wilson G. Ten quick tips for teaching programming. PLOS Computational Biology. 2018;14: e1006023. doi:10.1371/journal.pcbi.1006023

9. Wright AM, Schwartz RS, Oaks JR, Newman CE, Flanagan SP. The why, when, and how of computing in biology classrooms. F1000Research. 2020;8: 1854. doi:10.12688/f1000research.20873.2

10. Sweller J, van Merriënboer JJG, Paas F. Cognitive Architecture and Instructional Design: 20 Years Later. Educational Psychology Review. 2019. doi:10.1007/s10648-019-09465-5

11. Clark JM, Paivio A. Dual coding theory and education. Educational Psychology Review. 1991;3: 149–210. doi:10.1007/BF01320076

12. Steele-Johnson D, Kalinoski ZT. Error Framing Effects on Performance: Cognitive, Motivational, and Affective Pathways. The Journal of Psychology. 2014;148: 93–111. doi:10.1080/00223980.2012.748581

13. Wilson G. Teaching tech together: how to make lessons that work and build teaching community around them. Chapman and Hall; 2019.