Computing in the Classroom – Semester 1 Report
Exam No. B182394

Link to Portfolio: https://github.com/arbezy/CitC-Portfolio/tree/main

Introduction

In this report I will talk about my experiences teaching as a technical mentor at CoderDojo, specifically using and adapting the Raspberry Pi Foundation's Introduction to Python Pathway. I will then use the things learned from this material and from my own experience to produce a new Python exercise intended to be delivered in CoderDojo or a similar setting, although key design decisions for this exercise are included in a separate document.

CoderDojo

What is CoderDojo?

CoderDojo is a community of free local coding clubs for kids, they take place in public spaces, usually libraries and include "ninjas" aka students of ages 7-17 (although our Dojo is more on the younger side), and mentors to assist the ninjas (CoderDojo, 2023). My Dojo is smaller than a typical classroom with about ~14 ninjas, with ~8 doing Python and the rest doing Scratch, and as there is a team of mentors this means a mentor usually oversees a small group of 3-5 students, although there is no real set structure, rather a mentor helps where they're needed. Over the past semester, I have been volunteering at a CoderDojo in Edinburgh as a mentor, specifically for Python. This gave me chances to practise some of the teaching techniques covered in the course. I think this experience was invaluable as it's one thing to learn about a technique and another to practise it yourself to better understand where it works and where it may be less effective.

What practical pedagogies have I tried in sessions?

The pedagogy that I used most frequently was Teach Back. This is a deceptively simple pedagogy commonly used by healthcare professionals to verify a patient can make informed decisions or understand more about their condition and how to take care of themselves (Health Literacy Place, 2023). Research has shown it's effective in this environment in a study involving nurses teaching diabetics how to care for their diabetes (Negarandeh, 2012), although, I could not find any studies in a programming context. This would be an example of teacher-student teach back, but you also have variants of student-student teach back where a student will teach back a topic to another student. This is very effective for establishing a mutual understanding as it is formatted as a conversation, so in a teacher-student teach back, if the teacher is not satisfied with the student's explanation in their own words, they can rephrase or clarify some information and have the student teach it back again until the teacher is satisfied. Teachback is based on Glaserfeld's educational theory of "Radical Constructivism" (Herodotou, 2019), which builds off Piaget's theories of Constructivism. While Constructivism focuses on an individual making sense of a topic in terms of their experiences, Radical Constructivism is the more extreme version of this that all learning *must* be constructivist (Bergan, 2019). Glaserfeld suggests to teachers that rather than focusing on "sacred truths" they should give students opportunities to think, and a good way to stimulate this is through opening conversations with students, and getting them to verbalise their thought processes, which often brings inconsistencies in their understanding to light. Glaserfeld also heavily discourages teachers from telling students that they are wrong, but the emphasis here is on encouraging the effort they've put in to come up with a solution rather than avoid correcting misunderstandings (Glaserfeld, 2001). Therefore, in a teaching technique such as teach back a special effort should be made to recognise a

student's work, as the process could become discouraging if all the focus is on what is wrong with the student's explanations rather than what is right with them.

Another pedagogy I tried to use was Adaptive Teaching (also known as adaptive learning). This pedagogy stems from research suggesting that effective teachers should be adaptive in their teaching approach (Melissa, 2022), and thus should have agency to decide on lesson objectives, and the curriculum they teach (Duffy, 2005). It typically involves contexts with a range of mixed backgrounds like we have in CoderDojo, but it is also particularly relevant to educational inclusivity. Adaptive teaching can be split into macro or micro levels, where macro would refer to planning different lesson plans for groups of learners, and micro would refer to more individual-focused support, through ongoing observation and instruction (Hardy, 2019). Many examples of adaptive teaching in the modern-day focus on software to assess the knowledge of students (Munoz, 2022), which can then be fed back to a teacher, who can plan a lesson around the group's weaknesses, I would describe this as macro adaptive teaching, and not very suitable for CoderDojo. Instead, I focused on the more micro-adaptive teaching, trying to learn the styles and struggles of the group of ninjas I was supervising and trying to change my approach based on the individual. There is much less research on micro-adaptive teaching, but some would argue that it is much more commonplace and that practising teachers make these kinds of adaptions all the time (Corno, 2008). Not only this but several teaching frameworks will reference something along the lines of "Demonstrating Flexibility and Responsiveness" (Alvarez, 2011) without directly referring to adaptive teaching.

Finally, one pedagogy that I tried to implement was spaced repetition, I'm an advocate for spaced repetition and have previously used spaced repetition software (Anki) to assist in learning information for my degree. Spaced repetition (or spaced learning) has gained popularity after discoveries in neuroscience about how to produce long-term memories. It uses the idea of spacing learning across set periods and then coming back to that learned knowledge repeatedly to make long-term memories quickly. It has not just been tested on mice (Kramar, 2011) but has been used in a classroom setting to learn English vocabulary and shown to be efficient and effective at producing superior long-term retention compared to mass learning (Sobel, 2010), but no large-scale testing has been completed as of now. I thought that spaced repetition could be a good fit for CoderDojo as if we can equip the ninjas with the tools, i.e. knowledge of syntax and constructs quickly, then in sessions we can focus more on how to use these tools to become an effective programmer. Intuitively, spaced repetition could also be criticised for being too objectivist, filling a student's head with knowledge is only useful if a student knows how to use and apply that information, but studies have shown that spaced repetition does improve problem-solving (Kang, 2016), I think this is because now the knowledge is no longer a bottleneck so students are better equipped to solve problems set before them.

How well did they work?

Teach back came very naturally to a CoderDojo setting, where to ensure that a learner has properly understood the meaning of the code they have been writing, I can ask them to teach back a code snippet to me, and I can ask them to expand on what each part does or why we need certain things (a common example was why we need to cast an input to an integer in Python). This is quite useful as when following a tutorial like an RP pathway, students are eager to copy out the code and have a working program rather than to understand the code that they are writing. Teachback done this way helps slow students down to make sure that they take in the information they are being presented within the tutorial. In the session, I only used teacher-student teach back, as I only needed to supervise a small number of ninjas, if I was leading the session just by myself with no support it may be more suitable to have learners group up to do student-student teach back and then present back to me or another teacher together who can clarify their collective misconceptions. After every session I used teach back, I received good feedback from the students I supervised, and they seemed to be much more confident in their knowledge than when I didn't use it.

Adaptive teaching was not the most natural pedagogy to implement in my CoderDojo teaching, as I'm only in charge of the sessions every couple of weeks and this semester we have been completing a predefined course, so there is not much room for variation of the lessons meaning macro adaptive teaching was very difficult. However, I could still change my teaching approach depending on the ninja. Adaptive learning often includes using quizzes and assessments frequently to decide what to focus on, at the end of every activity in the Raspberry Pi Pathway there is a quiz, so I could use this along with seeing the common places ninjas got stuck and listening to the experiences of other mentors, to monitor which programming constructs or computational ideas learners struggled with. I could then follow this up by either taking some time during the lesson with an individual learner to go over a topic more thoroughly or if it was a more class-wide issue, I could try to explain that concept up at the front usually writing some Python on the TV for the students to see, and explaining as I went along. An example of when I did this was when I noticed that many students struggled to grasp the purpose of global variables, as they had no concept of local variables, so I wrote some code on the TV to try and better define them using some written code examples. I didn't just adapt the things being learnt but also tried to adapt to suit a learner's style, for example, one week when I was supervising two related ninjas on a project about designing a mask in Python: ninja A and ninja B, I found that ninja A didn't really care for the visual aspects of the program as they were there to learn to code, while ninja B was much more interested in making their creative visions come to life. I changed the emphasis of the support I gave each of them by providing A with more information on good coding practises and their importance in large applications, while for B I tried to focus on getting them to explore more design options in their program and show how Python could be used to achieve their visions.

The main weakness of adaptive teaching in my opinion is that it is hard to coordinate in a full classroom of 30 or more students. It's for this same reason that predecessors to adaptive teaching like "individualised instruction" from the 1970s weren't successful (Tomlinson, 2001). But in a CoderDojo context, we don't really have this issue as mentors are usually assisting the same students close to them, and because there are several mentors this divides the class down into much more manageable numbers, to the point where the quizzes are not that useful as you can observe the difficult topics as students get stuck on them. Although, as mentioned before CoderDojo doesn't have a lesson plan like a classroom would have so core material being taught is usually out of a mentor's control. In our Dojo, I think adaptive teaching worked well, but there is a limitation to how much adaptive teaching can be used due to volunteers' time constraints; all other volunteers are working full-time jobs (aside from me!) so don't have time to plan out lessons that have been handcrafted to support the ninjas.

To use spaced repetition, I would see where in the tutorial the learner was taught some relevant information like how a function call always requires brackets even if there were no parameters, I would then wait some time and during the class come up to them again and ask them how to call a function. I would do this about 2-3 times during the session. Unfortunately, I think spaced repetition did not work well during the sessions, and after the week I trialled it, I never used it again. I found that it was more prohibitive than it was encouraging, this is because I felt like it detracted from the constructivist setting, in which learners are creating personalised programs, as it became a very objective way of learning, where you're either right or wrong and there is not much room for creativity. I don't think all spaced repetition is this way, rather my usage of it in this setting. I think that I would be much more effective in this environment if I planned the session from scratch, where I could split the activity into 3 chunks each containing uses of all the programming constructs learning objectives for that session. Because we are following a pathway the session can't really be run this way and so, in my opinion, spaced repetition doesn't work very well. Another reason this might not have worked very well could be related to "cognitive overload", by interrupting ninjas as they had

moved on to other tasks, this split their focus from what they were currently learning to what they had previously learned, so they have to manage both sets of information at one time. It may have been wiser to wait for moments when they were between tasks before asking them to recall something, as this would reduce the cognitive load.

I've written about some of my other experiences teaching, and some of the related theories in an Appendix at the end of the document, to keep the report shorter.

Conclusion

In conclusion, I found that with small groups with learners already keen to learn that teaching using social constructivism worked very well and was a very natural fit for this setting. Particularly the more experienced students were very interested in not just absorbing the teaching but learning from the experts' examples of how to think like a computer scientist. I found that this style had noticeable benefits on how well students solved tasks, but also on their confidence that they were now more equipped to solve problems independently. I also learned the importance of engaging students by letting them take ownership of code by enabling personal design decisions. Finally, through my experiences with spaced repetition, I learned that the implementation of a pedagogy is important and that no technique is a magic bullet that will work with no consideration of the environment.

Future Work

In the future, I want to encourage not just having one source of expert knowledge from which the students can draw from but to encourage students to work together to improve each other's knowledge, which I felt was a dimension that I was missing from our sessions. As well, I think with some better planning I could create a session using spaced repetition, that would be much more suited to the CoderDojo setting.

Appendix - Other Experiences

There were several experiences and techniques I learned while mentoring CoderDojo that weren't directly inspired by reading a pedagogy (but usually still related to them). In this section, I'll list some of the ones I found the most formative to my teaching style and the theories associated with them.

It is well known that feedback is one of the most powerful influences on learning (Hattie, 2007), and this is especially true when using teachback. This is a huge topic with lots of research that I won't cover here, but it's useful to note that the feedback I'm giving to learners is less directly evaluating how well they are working presently and more encouraging them to think about what they've done so far and what they will do next. The main lessons I learned from this was that learners feel more engaged when they take pride in their work, and the easiest way to get them to take pride in their work is to make it meaningful to them through personalisation, and the resources we use lend themselves very well to this. As well, I tried to have students reflect on the progress they've made at the end of a session, which the resources also worked well for as they had visual elements so usually the ninja would have a finished project that they could show to their parents or peers (although the learning materials did hinder this somewhat as they required an account to save code, and many young ninjas are not good at remembering passwords!). I also learned that giving good feedback and learning when to give feedback is a hard skill to master, so in the future, I may try to research feedback frameworks to help with this.

In class, we covered Maton's semantic waves, which is a way to express the semantic gravity and density of an explanation over time. Semantic gravity relates to the degree the explanation relates to context and semantic density refers to the degree how much the explanation focuses on sociocultural practises. In a semantic wave graph these are inversely proportional with high-density, low-gravity explanations usually involving abstract and complex, very contextual meanings. While low density, high gravity would mean simpler meanings often using some socio-cultural analogy (Maton, 2013). From this, I learned the importance of avoiding semantic flatlines, where you never repack a concept after unpacking it using a metaphor or analogy. I found this very valuable in CoderDojo, particularly when it came to explaining programming concepts like variables and abstraction, but also very useful for handling jargon i.e. "calling" functions which came up in a session.

On the 13th of November, I led CoderDojo and ran into some trouble regarding ninjas working at different paces. This is not unique to CoderDojo, but I feel could be more severe than a normal classroom environment as we have a larger range of ages, so younger students tend to need more support than older ones, also some ninja's parents work in software development, so they have usually had more experience than other learners. In response to this, I tried to set extra challenges for these advanced students, the pathway we used didn't have much of a focus on loops these weren't really setting them new features to implement but rather having them try to understand a function better by asking them to change its parameters to achieve a certain effect, such as making the trail from our rocket from the lesson longer or denser. Another way I found to do this in sessions I didn't lead was trying to get them to include a loop somewhere in the project. It was doing this that I found out a very engaging challenge for ninjas was having them stress test their machine by using very large numbers or infinite loops, and they usually got a lot of enjoyment out of bricking their machine. After this session I advocated for not having a leader role with everyone following along at the same time but instead for learners to work individually with a mentor supervising a small group as I thought this would also help alleviate this problem. I only realised this later, but I think my solution to this problem would fall under the adaptive teaching that I previously discussed.

In our Dojo, it is up to the learners what they want to do, and they can choose to opt into the Python or Scratch groups, but some will choose to do CodeCombat or some other programming language. For

these students I tried to focus on computational thinking, there was a particular week when a ninja was working on a unity project and following an online tutorial. I do not know Unity so instead had a discussion with him about a bug in his code that let a player jump mid-air. We discussed the logic you could use to try and stop this from happening, as well as how this would change if you wanted to implement a double jump. This also draws on Vygotsky's theories of social constructivism, that mediated activity, in this case verbal interaction with a more knowledgeable other, promotes greater cognitive development, in this case learning how to dissect a problem and how to engineer a solution. I was very happy with the outcome of this, as all programming uses some kind of computational thinking, so it gave me the confidence to try this in the future for similar situations. As well the student seemed to get something out of the session that he wouldn't have if he had just completed the tutorial at home.

As the Raspberry Pi Intro to Python Pathway has been produced by some knowledgeable people, there is almost always room for personalisation in the projects. However, some ninjas tended to skip these sections always choosing to pick default options, and just copying code with little engagement. In response to this, I tried to encourage ninjas to take control of the program and make it theirs by changing aspects of the code, such as the colours being used or changing the values of constants to make their games harder for example in one session we made an "endless runner" style game, and I encouraged some ninjas to go off script and change the number of obstacles and the speed of the game. A relevant framework to this is Use-Modify-Create (Lee, 2011), in particular Modify, which is something that the makers of the Raspberry Pi pathways cite themselves in their creation of these materials. Modification not only engages students but also encourages new skills and understanding as they learn how to fulfil increasingly sophisticated modifications. Use-Modify-Create (UMC) has shown to be successful in computational thinking in an A/B study of UMC in middle schools in the US (Lytle, 2019) which found students not using UMC found it more difficult in the beginning to jump straight to creation, and also that the UMC felt significantly more ownership over their final code, despite both groups completing the exact same task.

The last main thing I tried was to not give too much help to students. I tried to be strict about how much I intervened when a ninja had a problem, this meant not taking the keyboard away from them and correcting their errors but trying to guide them to either see the solution themselves or to at least have them correct the errors themselves for more tricky ones. This is an example of guided learning (Billet, 2012) which is deeply rooted in social constructivism. I thought this would help students in the future spot errors and debug their own code, which I think is an undertaught skill compared amount of time a more experienced programmer will spend debugging in real life. It was also brought to my attention by one of the students that he wasn't just trying to learn to program but also how to type faster, so he made sure to type everything out and not copy and paste. This prompted me to think about how I could teach some more realistic programming skills, so I also tried to teach some shortcuts common to programming, such as commenting and unindenting blocks of code.

Usage of AI

I used Grammarly to assist in grammar-checking my report and making it a little bit clearer and more concise.

References

(CoderDojo, 2023) CoderDojo Home Page: https://coderdojo.com/en/. last accessed: 7/12/23

(Tomlinson, 2001) Tomlinson, C. A. (2001). How to Differentiate Instruction in Mixed-Ability Classrooms. 2nd Edition. Chapter 1.

(Herodotou, 2019) Herodotou, C. et al. (2019) Innovative Pedagogies of the Future: An Evidence Based Selection. Frontiers in Education, Volume 4. DOI: 10.3389/feduc.2019.00113

(Glaserfeld, 2001) Von Glaserfeld, E. (2001). Radical constructivism and teaching. Prospects, 31(2), 161-173. DOI: 10.1007/bf03220058

(Negarandeh, 2013) Negarandeh, R. et al. (2013) Teach back and the pictorial image educational strategies on knowledge about diabetes and medication/dietary adherence among low health literate patients with type 2 diabetes. Prim. Care Diabetes 7, 111-118, DOI: 10.1016/j.pcd.2012.11.001

(Health Literacy Place, 2023) Educational website including a video on using teach back to promote health literacy: https://www.healthliteracyplace.org.uk/toolkit/techniques/teach-back/, last accessed: 6/12/2023

(Bergan, 2019) Van Bergen, P., Parsell, M. (2019) Comparing radical, social and psychological constructivism in Australian higher education: a psycho-philosophical perspective. Aust. Educ. Res. 46, 41–58. DOI: https://doi.org/10.1007/s13384-018-0285-8

(Melissa, 2022) Melissa A. Gallagher, Seth A. Parsons & Margaret Vaughn (2022) Adaptive teaching in mathematics: a review of the literature, Educational Review, 74:2, 298-320, DOI: 10.1080/00131911.2020.1722065

(Duffy, 2005) Duffy, G. G. (2005). Developing metacognitive teachers: Visioning and the expert's changing role in teacher education and professional development. In S. E. Israel, C. C. Block, K. L. Bauserman, & K. Kinnucan-Welsch (Eds.), Metacognition in literacy learning: Theory, assessment, instruction, and professional development (pp. 299–314). Mahwah, NJ: Lawrence Erlbaum Associates.

(Hardy, 2019) Hardy et al. (2019). Adaptive teaching in research on learning and instruction. Journal for Educational Research Online, 11 (2) (2019), pp 169-191

(Corno, 2008) Corno, L. (2008) On Teaching Adaptively, Educational Psychologist, 43:3, 161-173, DOI: 0.1080/00461520802178466

(Alvarez, 2011) Alvarez, M E. (2011) Danielson's Framework for Teaching. Children & Schools 33(1):61-63. DOI:10.1093/cs/33.1.61

(Munoz, 2022) Munoz, J L R. et al. (2022) Systematic Review of Adaptive Learning Technology for Learning in Higher Education, DOI: 10.14689/ejer.2022.98.014

(Hattie, 2007) Hattie, J., & Timperley, H. (2007). The Power of Feedback. Review of Educational Research, 77(1), 81-112. DOI: https://doi.org/10.3102/003465430298487

(Maton, 2013) Maton, K. (2013). Making semantic waves: a key to cumulative knowledge-building. Linguistics and Education, Volume 24, Issue 1, 8-22, DOI: https://doi.org/10.1016/j.linged.2012.11.005.

(Kang, 2016) Kang, S. H. K. (2016). Spaced Repetition Promotes Efficient and Effective Learning: Policy Implications for Instruction. Policy Insights from the Behavioral and Brain Sciences, 3(1), 12-19. DOI: https://doi.org/10.1177/2372732215624708

(Lee, 2011) Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. (2011). Computational thinking for youth in practice. ACM Inroads 2, 1 (March 2011), 32–37. DOI: https://doi.org/10.1145/1929887.1929902

(Lytle, 2019) Lytle, N., et al. (2019) Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. ITiCSE '19. 395-401. DOI: https://doi.org/10.1145/3304221.3319786

(Billet, 2012) Billett, S. (2012). Guided Learning. In: Seel, N.M. (eds) Encyclopedia of the Sciences of Learning. Springer, Boston, MA. DOI: https://doi.org/10.1007/978-1-4419-1428-6_34

(Kramar, 2011) Kramar, E A. et al. (2011) Synaptic evidence for the efficacy of spaced learning. Proceedings of the National Academy of Sciences. 109 (13) 5121-5126. DOI: 10.1073/pnas.1120700109

(Sobel, 2010) Hailey S. Sobel, Nicholas J. Cepeda, Irina V. Kapler. (2010) Spacing effects in real-world classroom vocabulary learning. Applied Cognitive Psychology, Volume 25, Issue 5. p763-767. DOI: https://doi.org/10.1002/acp.1747