TEACHING THE MIDDLE SCHOOL PROGRAMMER

An Ohioan's Approach

Blade Frisch

Teaching the Middle School Programmer

An Ohioan's Approach

Blade Frisch

This book is for sale at
http://leanpub.com/middleschoolprogrammer

This version was published on 2018-03-30

Tweet This Book!

Please help Blade Frisch by spreading the word about this book on Twitter!

The suggested hashtag for this book is #MiddleSchoolProgrammer #AnOhioansApproach.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#MiddleSchoolProgrammer #AnOhioansApproach

This book is dedicated to the CSN Faculty, without whom this book would never have been written.

Contents

Chapter One: Preparing Yourself

The first step to approaching any task is to prepare yourself. You don't play a concerto without warming up or drive from Chicago to New York City without packing some bags. Preparing yourself is the first, and arguably most critical, step in this book. If you read nothing else, at least finish this first chapter. Once you have prepared and shaped yourself for the job, it will be much easier for everything else to fall into place. Let the journey begin!

Gaining Content Knowledge

What do you do if you are not a programmer?

I have encountered several teachers who don't have a degree in computer science and haven't programmed in nearly any capacity, yet they are roped into teaching a programming class. If you find yourself in this position, I have only two words for you: don't panic. I say that because, in today's world, there is so much information available in so many ways (something you will probably teach on!) that you are able to fill in a large portion of the content knowledge with a little hard work, dedication, and WiFi.

The first and most effective option available to you is to take a programming class. Perhaps your school district will pay for you to audit a class at the local university, where you can learn in a classical environment that many people learn to program in. If this option is not available to you, there are also several online resources of high-quality. Universities such as Stanford and MIT offer many of their classes online and free of charge[1][2]. Google offers an in-depth

[1] https://online.stanford.edu

[2] https://ocw.mit.edu/index.htm

guide for growing your technical skills[3]. E-learning companies such as Udacity and Udemy offer high-quality courses on a variety of subjects[4][5]. These are all resources that I use almost every day to continue to grow my own content knowledge, and so should you. The learning journey never ends in the technical world.

Another important task to undertake is getting familiar with the software world. Being able to program is only a portion of the job; knowing what to do with that skill set is the other side of the coin. In an ideal world, you would spend some time "in the industry". This could happen as an intern, in a shadowing type arrangement, or even simply by taking part in an online community. Now, you may be thinking, "I'm teaching middle school. Will I really be teaching them about how to be an adult software engineer in the sixth grade?" The short answer: absolutely. The long answer is that you always want to have a goal in mind, and, in this case, the goal is training good programmers. You can have approximations of the real world at all levels, including middle school. Instilling these good practices and ideas from day one creates a lifestyle rather than something they learn and never

[3] https://techdevguide.withgoogle.com
[4] https://www.udacity.com
[5] https://www.udemy.com

use again after the assignment or class is over. In addition to trying to get some time in the industry, you also need to try and get/stay familiar with current events and technologies. The tech world is always changing, and at a rapid pace. Read articles about tech events, listen to podcasts, watch videos (there are many great TED Talks, and YouTube has a variety of tech channels). Follow companies like O'Reilly Media and Code.org on your favorite social media platform. Finally, you can use a technique that I am infamously known for: cold-calling. As I am researching a topic, I will often come across names and/or companies that work in this field. I will call or email them and try and start a conversation. In this vein, contact other schools and universities, talk to the teachers/professors, find out their favorite resources, and make connections. Making connections and finding a community will give you an incredible support network and safety net.

Now that have some skills and resources to fill in your knowledge gaps, it's time to get your hands dirty. This is a theme you will find throughout this entire book. I am 100% convinced that you can't learn, or teach, programming without getting your hands dirty is some code. Learn to code by writing code. You can find many programming challenges online, with some popular ones being CodeChef, HackerRank, and Top-

Coder[6][7][8]. These websites host competitions, but they also have many practice problems sets in a variety of languages on a variety of topics. Working your way through these sample sets will help build your mental models, algorithmic bank, and programming skills. Once you have tried writing some programs on your own, move on to analyzing the work of others. GitHub is the leading code-sharing platform, hosting code from complete amateurs to the Linux kernel source code[9]. You can browse by topic, language, and collection, search users and projects, and share your own code. Find a project in your language of choice and begin to pick it apart. Something I do when I encounter a new project is I will print out the code, sit down with a pencil and highlighter, and begin to annotate. I will highlight important bits of code and write down what they do or questions I may have, bracket off key words, and draw lines connecting segments of code. This helps me to develop a deep understanding of what this code is doing, including what the job of each gear in the system is. The final step in this process is taking someone else's code and modifying it. This can mean altering its current functionality or extending it. This could even be the same

[6] https://www.codechef.com
[7] https://www.hackerrank.com
[8] https://www.topcoder.com
[9] https://github.com

program that you chose from the previous paragraph. This gives you practice in understanding code that you didn't write and writing code yourself in the same project. I give you this challenge because, in the industry, you are more often than not working with code someone else wrote rather than writing everything yourself. Code analysis is just as important as code generation in these circumstances.

I give the non-programmer one final note in this section about content knowledge: this is a never-ending process. Just as a scientist never stops conducting experiments, the musician never stops practicing, and the author never stops writing, so too must the programmer never stop programming. Staying current on the news, writing code, picking apart other's code, modifying projects, these are all daily aspects of the programmer's life. While they may not be the life of a middle schooler, it could be someday, and you need to be the model.

What to do if you are a programmer?

If you are someone who has a background in both teaching and programming, you are in a sweet spot. You can still gain something from reading this book

(at least that's the hope), but you can skim this section. If you are a programmer with no primary or secondary teaching experience, this is for you and you probably shouldn't skim this section.

The first task for you, and potentially the hardest, is learning to forget…or at least give the illusion of forgetfulness. For the programmer, prior knowledge is as much a curse as a blessing. You are able to look at a problem, pick it apart, see code segments flash before your eyes, incrementally build a solution, and test your code as easily as making eggs and toast. In my observations, and I am guilty of this, we forget these are skills we learned over many years the middle schooler has not gone through. Middle school students still need to build this skill set, and you, as their teacher, cannot assume they have it in your class. You may be asking yourself, "What on Earth does that mean?" This means that you must walk through all fifteen steps you skip. You have to explain every task your brain does automatically rather than write the steps on a board and assume the students can see the connections. You will have to repeat, redemonstrate, and reteach the skills, concepts, and programming idioms more times than you will enjoy. However, taking the time to do all of this ensures your students will build up these skills and processes until they become second nature. This may take weeks, months,

years, or maybe forever, but you must be just as diligent in learning to forget. As a wise Jedi master once said, "You must unlearn what you have learned."

The next task is to choose appropriate topics. The next chapter will take a much deeper look into this idea, so I will be brief. You must choose topics that are developmentally appropriate for the middle schooler, and how to explain higher level concepts that can't be skipped in an appropriate way. For example, you don't want to be teaching middle school students about the fine details of how programming languages are made, such as a compiled versus interpreted languages or strongly versus weakly typed. These are topics that are beyond middle schoolers and not developmentally appropriate. Middle school students also don't need to know that arrays/lists are sequential memory locations that use a pointer to move from index to index. However, they will need to know that arrays/lists allow you to group data together under one label. Spiraling your curriculum, and this spiraling can occur across grades, will ensure those details are touched upon eventually[10].

Finally, you will need to gain pedagogical knowledge. In order to become an effective teacher, you must study teaching and learning in some aspect. The best

[10]Look up Jerome Bruner's Spiral Curriculum ideology.

solution is to audit some education classes at a local university. If you are able, take educational psychology, classroom management, and child development. In my opinion, these are the most critical subjects for a teacher to know. Sometimes this is not possible, so to try to find and attend professional developments and/or workshops on these topics. Since these are the core pillars that support the teacher, there are many opportunities all around the country. Another great tactic for gaining knowledge is to observe other teachers. Before I became a programming teacher, I taught middle and high school orchestra. I had worked with middle school students for many years and felt quite comfortable in that environment. However, when I started teaching middle school programming I realized that programming classes are a completely different beast with its own set of challenges. My first month or so I had a lot of difficulties in the classroom, so I talked with my colleagues and set up observations. I would go to their classrooms, which often had some of my students, and observe how they taught and managed their classrooms. I was able to learn a lot from this, and it gave me plenty of new tools for me to implement in my own space. After this, have other teachers come and observe you. This is also something I did my first year teaching middle school programming. I asked my colleagues

to come in, observe, and then meet afterward to talk about what they saw. Having someone who is both objective and subjective, meaning they aren't teaching the class but they still know you and teach the same students, allows them to point out things you may not be seeing and help you better yourself. This also promotes a sense of camaraderie between you and your colleagues, showing that you are invested in the school and students and care about the expertise of your fellow teachers. Finally, make sure you take time to read books and articles. You will never know all that there is to know about teaching. Don't take that as an insult, but an inspiration. There is always more to learn, which means you can always get better. Read articles and blogs about classroom management, read books by great teachers and learn from what they have done, continue to expand your knowledge base and tool belt. The world, and therefore your students, are ever-changing, and you must keep current in order to effectively teach and reach them.

What I wish someone had said to me

In this next section, I simply want to share a few things that I wish someone had said to me when I was a new teacher. So often there are life lessons that you learn along the way that simply can't be learned through study, but that doesn't mean you must be caught off guard. Almost every teacher can tell you a dozen stories for each of these maxims, so take these to heart and prepare to go through your own lessons.

It's okay to fail

I have a perfectionist personality. If I am going to do something, I am going to do it right. This can serve the programmer and teacher well, as long as you recognize that failure is also an okay option. Sometimes you will fail a student, another teacher, or your administrator. They will look to you for something, unrealistic or not, and expect you to deliver. Sometimes you can't. You will try offering a new class or teaching a new topic, and it goes miserably. The students don't understand, your lessons and assignments aren't making sense, and it fails. This can and will happen to you at some point. There is no escaping

that reality. What you can change, however, is how you respond to it. For every failure, take the time to reflect. Ask yourself what went wrong, why it went wrong, should you try again, and what do you need to change to make it work next time around? Failure is a great teacher, and that is something you need to instill in your students as well. Work them through their failure, ask them these same questions, and help them to become a reflective and self-aware person.

You will not reach everyone

The goal of a teacher is to reach their students. I think that every teacher has a desire to change the world, and they want to make sure every student leaves their classroom better than when they entered. Unfortunately, you will never reach all your students. There will be students who have it in their mind they will not enjoy or participate in the class. Do your best to convince these kids that it is worth their while to be an active student, but they may have their mind set. Sometimes people, including middle schoolers, don't want to change. You may also simply not click with a student. They are not a bad student, you are not a bad teacher, but there just isn't a connection. There is never a one-size-fits-all, and that applies to teachers as well. Do your best to make a connection, but be

aware that you may not be the inspiration teacher for that student, the one they will always remember as making a difference in their lives. And that's okay. You may not harvest the crop, but that doesn't mean you should stop planting seeds and watering the field.

Bad Lessons Happen

Every class can't be a perfect moment of pure inspiration. There are days where the lesson will be average at best or even flop. Everyone has off days, and that's okay. I have had days where I tried to teach something and my students are simply not understanding. In these moments, I would be honest with my students. I would say to them, "Guys, you don't seem to be understanding this the way I am teaching. What can I do to make this make more sense?" While this may not lead to the most profound discussion, including the students and trying to work with them helps the students to feel like they have some ownership of the class. After this talk, rework the lesson and try again the next class. You can always make things better or try to teach in a new way.

Listen to your students

Like I mentioned in the previous maxim, take the time to listen to your students. When you involve the

students in their learning process, they are much more willing to buy into the class. Giving them a chance to have their voices heard, take leadership, and/or show off can make a huge difference. In my classroom, I will have "student teachers" on occasion during the review time, where they lead the review of the previous class's material. This gives students a chance to shine, be the leader, and put themselves in my shoes. Once you have established that environment and tone, students are more than willing to work and work with you.

You are now ready to move forward. Even if you haven't done everything in this chapter, you are at least aware of the things you need to do. This may be a good place for you to pause your journey through this book, reflect, and work. Take some time to attempt these tasks. Think on the maxims. Grow and prepare yourself, and then come back here and resume the journey.