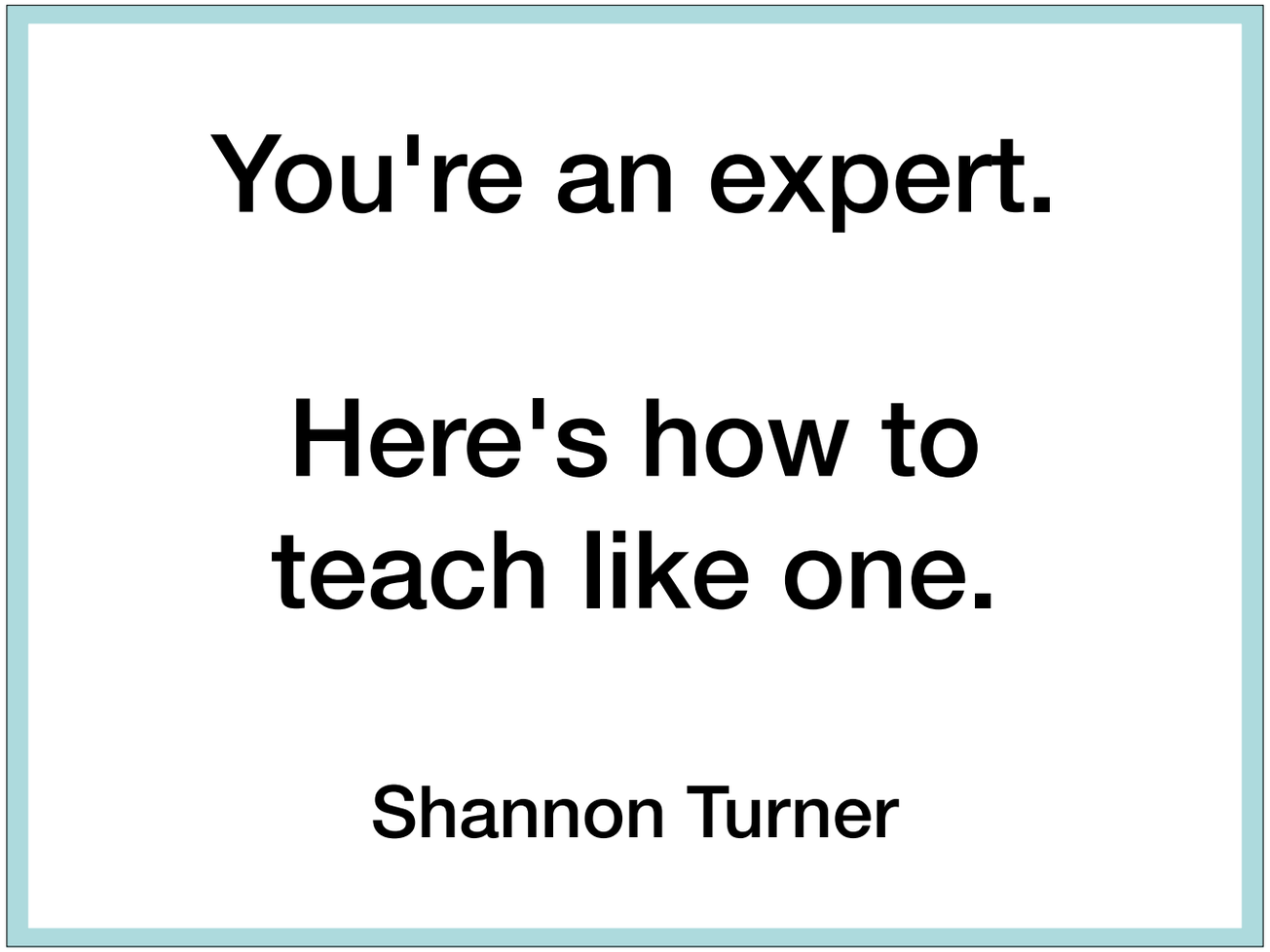


Knowing how to code and being able to teach it are two separate skills. When we have expertise in a subject, it's common to take for granted that we'll be able to effectively communicate our expertise to someone else. You'll learn (or re-learn!) how to teach and discover practical examples you can put to work right away.

By sharpening your teaching skills, you'll be a more effective mentor, trainer, and team member.

Who am I?

- Founder, Hear Me Code
- 3500 students
- 200+ teachers and teaching assistants

Hear Me Code is an organization I started that offers free, beginner-friendly coding classes to more than 3000 women in the Washington, DC area. I started informally teaching a group of friends around my kitchen table and never expected Hear Me Code to be a "thing" but just four years later we've grown to over 3000 women all learning Python together, for free. I developed the curriculum and for the first eight months, I taught every single class, but now I hardly ever get to teach because we focus on turning students into teachers.

Teach like an expert

- Know your audience
- Use examples that connect
- Know what to hold back
- Foster a love of learning

This presentation is based on everything that I've learned over the past four and a half years in building Hear Me Code. I run one-on-one train-the-trainers with every teacher, I've taught dozens of times, and when I'm not teaching, I'm serving as a teaching assistant. Here are the four main categories of advice I would give to help you be a better teacher.

Know your audience

Here, I'm going to take my own advice -- who's going to get the most out of this talk? Do I have to be a teacher? No.
No previous teaching experience is required, but this talk can benefit anyone who trains, mentors, or teaches others (formally or informally), especially:
* Someone who mentors a junior developer and wants to help them reach their potential
* Someone who manages mid-level developers
* Someone who communicates technical information to non-developers
* Someone who trains developers or teaches coding at any level
After the talk, you will have practical takeaways to help you more effectively communicate with developers and non-developers alike.

Connect with your audience

Why should they listen to you?

What can you offer them?

You might have noticed my very first slide was: who am I, why might I be an expert worth listening to on the subject of teaching?
Don't just jump into your lesson! Introduce yourself -- you'll be spending a lot of time together! I also take attendance with every class, because I care about getting to know them, too.

What motivates them?

Know why your audience wants to learn
can help you tailor your lesson

And when you have a large class size, tailoring your lesson means touching on the motivations for many different groups of people in the room.

This room is a great example of a large class size -- and notice how I framed this talk as being interesting and applicable to many different groups of people -- formal and informal teachers, managers, mentors, and so on.

Set expectations

What is the goal for your lesson?

What do you want students to take away from your lesson?

Students should know up front what they will be learning from your lesson. You should have a plan for how you're going to meet your lesson goals. Your goals should be obvious and explicitly stated up front.

For example, I stated my goal for this talk is to convey a series of practical tips that you can put to use to help you be a better teacher, mentor, manager, and communicator.

Empathize with your audience

What is their level of understanding?

Don't assume!

Assume too high, and what you're teaching will go over people's heads.
Assume too low, and you're probably being condescending.

You can ask people, but also be mindful that people may not want to be 100% truthful about their level of comfort with the material and may pretend they know.

What's the solution? Re-train the way you teach so it's accessible to people at all levels.

Don't make anyone feel "less than"

Nobody likes being talked down to

This is a big one -- if people feel like you're talking down to them, they'll shut down. They'll get a bad vibe off you. And bottom line -- you will be less effective. Everyone loses.

Remove distractions

What might get in the way of learning?

Think about distractions in the broadest sense possible. Things like: sexist jokes, unexplained acronyms, environmental things like noises, I could go on.

Anything that could interrupt the learning process.

In Hear Me Code, we make sure that students have all of the slides downloaded so they can refer back to them during the lesson and the teacher is free to switch screens without worrying about students interrupting their flow with "Could you put that back up on the screen?"

What could get in the way of learning? Remove these roadblocks! And if you're concerned about getting these slides I'm using now, I'll make sure everyone has a link to get these at the end of the talk.

Use live coding sparingly

Live coding can be difficult to follow
and can derail your lesson plan

A pinch of live coding here and there is great. But be careful that it's not your only method of teaching. It can be difficult to follow -- students have to pay attention to your screen and type on theirs at the same time -- and it's difficult to refer back to a known, working example.
If you're only using live coding, you're going to cover less ground than you think. You'll be prone to making errors that will take you off-script explaining how to fix them, and it's easy to get away from your lesson plan -- you might not end up meeting your lesson goals.

Remember your journey

Where are they in their journey?

Nobody hits a home run the first time.

This is related to "empathize with your audience" but it deserves its own slide.
You didn't become an expert overnight.
Remember that this was difficult for you once, too.

Smash jargon

Define acronyms and insider speak

Better yet, use more accessible language

If you're using acronyms or insider-speak, make sure you define them rather than assuming everyone knows what they are.

Keep it conversational

Speak as plainly as possible

"Hey Python, can you open this file?"

Use uncomplicated words to explain what you mean.

When explaining concepts, it can help to imagine yourself having a conversation with Python.

Hey Python, can you open this file?
    Sure Shannon, which file?
attendees.csv
    Okay, are you reading this file or writing to it?
Reading from it.

Know your audience

- Connect with them
- Find out what motivates them
- Set expectations
- Empathize with them
- Don't make them feel less than

Know your audience

- Remove distractions
- Use live coding sparingly
- Remember your journey
- Smash jargon
- Keep it conversational

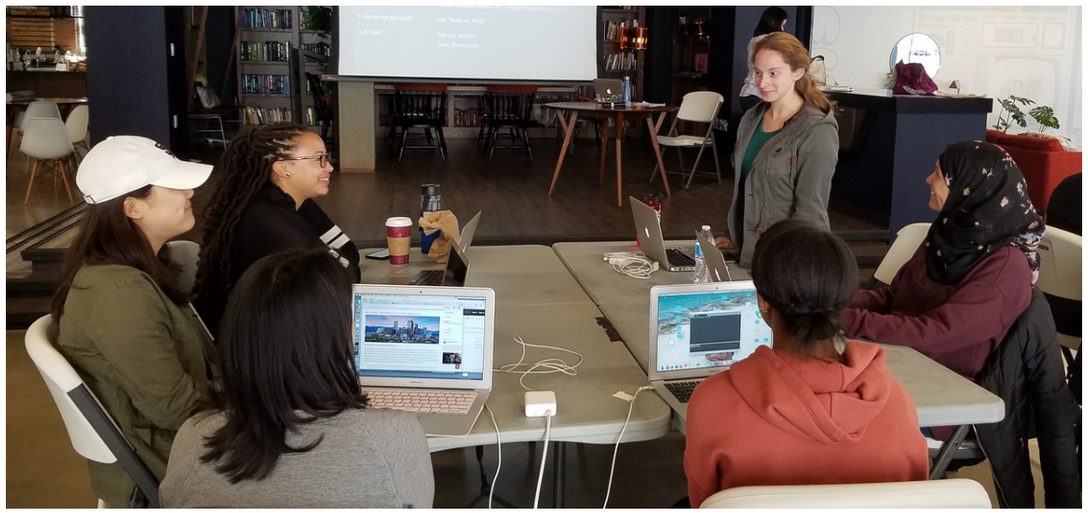
Examples that connect

Use examples people already understand

At Hear Me Code, we take attendance to teach **for** loops and again to teach **enumerate( )**

We've all taken attendance many times in our life. One way that can look is stand up, say your name, sit back down. And we actually do this in Hear Me Code to demonstrate how a for loop works.

Use examples people already understand



(A photo of people standing for attendance at class)

Folks stand up, say their name, sit back down. The code for this would be: for name in attendees: print name. This is really how we teach for loops!

Other good examples for lists and for loops: days of week, months of year. Use things people already understand!

Use specific examples

Using foo/bar/baz are less helpful than any specific, concrete example

Remember to use examples that people already understand!

Foo, bar, baz - these are all abstract generalizations. They don't mean anything. They also break important rules we teach beginners like "use meaningful variable names."
Any concrete example is better than foo/bar/baz, even if it's not directly applicable to a learner's exact situation.
Some of you may be comfortable with foo/bar/baz, but new learners really struggle with that!
I know we use foo/bar/baz because it's quick. But I want to challenge you to use specific examples that people already understand. It will take you extra time. But it will be worth it, I promise.

Be practical

How will students be able to use this?

Don't just teach syntax, teach the context

At Hear Me Code, we demo real (skill-level appropriate) projects at the end of class.

One of my favorite demos is at the end of lesson one on strings and conditionals, one of our frequent teaching assistants shows how to use string methods, slicing, and conditionals to tell whether a social security number is valid. And you can do this by the end of lesson 1.

Another way of thinking about this: **don't just teach syntax, teach the context** for why you would use this code.

This is a great way for you to draw on your expertise. What's a practical situation or example when you've used this concept in your work?

Why? So what?

When will I use this?

These questions will help clarify if your examples are practical.

These questions will help clarify if your examples are practical for one, and two, if you should be teaching this concept at all. Or right now.

And your examples don't have to be applicable to every situation. But specific examples where it's clearly understood **why** something matters or **how** you're going to use something are easy to grasp.

And maybe ask yourself: is this topic right for my audience right now, or is this something I should save for later?

One concept at a time

It's challenging to learn more than one concept at a time.

If you're using slides, be careful not to have too much on your slides! But this is more than just about making sure your slides aren't cluttered -- be careful to only teach one concept at a time or it's easy to get overwhelmed and not learn either concept very well.

In an old version of my lesson two slides before I took my own advice, I was teaching the three main ways you could use list.pop( ) all on a single slide. -- I had my list of days of the week, and had days_of_week.pop( ) to remove the last item from a list, days_of_week.pop(index) to remove a specific item from a list, and day = days_of_week.pop( ) to remove an item and save it to a variable.

That was all on a single slide - what was I thinking? Now it's broken up across three slides, and each concept is much easier to grasp because students are learning them one after the other.

Model good behavior

When providing sample code, use meaningful variable names.

Think about all of the best practices we impress upon people and immediately discard as soon as we're helping someone learn -- and I'm not just talking about meaningful variable names! Is your code well-documented? Is your code clever or is it readable?
Model that behavior that you want to see.

Be flexible

Great examples connect with most people, but it's rare that one size fits all

Be flexible. You might need to teach something multiple times, multiple ways, especially if you have a large class. You might need to try different approaches. Your inner dialogue might say, "Hey Shannon, aren't you just re-stating the same thing in different words?"

Well, yes. But it can be helpful!

Mistakes happen

When you make a mistake, explain what happened and how you're fixing it

Turn your mistakes into teachable moments! If you make a typo, narrate what happened.

"Hmm, Python is giving me a NameError here, telling me I haven't defined this variable. Oh! Oops, I misspelled this variable name! There, now it's fixed."

Do it wrong on purpose

When your students get errors, will they be prepared?

Sometimes, doing it wrong on purpose can actually clarify what's happening.

At Hear Me Code, when we teach strings, we show how you can use single quotes and double quotes interchangeably -- as long as you're consistent. But what happens if you start your string with a single quote and end it with a double quote?

Making these mistakes on purpose means we have a chance to walk through the error message -- and the next time students encounter it, they'll have a better sense of what to do.

Examples that connect

- Use examples they already understand
- Use specific examples
- Be practical
- Ask yourself "Why?" and "So what?"
- One concept at a time

Examples that connect

- Model good behavior
- Be flexible
- Mistakes happen
- Do it wrong on purpose

Know what to hold back

Answer the question asked

Offering lots of extra information or going on a tangent can be confusing

This is related to the rule "one concept at a time" -- if you're offering extra information, going on a tangent, going too far off-script / ad-libbing too much, it's very easy to end up being more confusing than enlightening.

Nuance isn't always helpful

Teach the most common situations

That painful corner case you've bumped up against? They might never encounter it.

When teaching slicing in lesson 1, we start by teaching the start and stop parameters. Yes, there's also a step parameter you could use to get every other slice -- but it's used much less frequently, so we don't even mention it.

If a specific question comes up, like "what if I wanted to get every other character in my slice?" I usually say, "there's a way to do it but it's less commonly used. But since you asked, here's how."

Teach the pool, not the ocean

Know what is relevant in the moment
and what is best saved for later

This is related to "nuance isn't always helpful" and "answer the question asked" but I think this deserves its own slide.

Learning is like an ocean -- impossibly vast, deep, murky, lots of pressure, and filled with sea monsters. That's why when we're first learning to swim, we start in a pool.

Don't overwhelm new learners with the ocean of information that they will *eventually* learn -- teach the pool, not the ocean.

Don't cover too much material

People have limits.

Is your lesson plan realistic?

Remember that the material you're teaching probably feels like second nature to you by now. It's brand new to your audience. It might take them longer to learn it than you think.

The most common mistake I see is teachers overestimating how much ground they'll be able to cover in their tutorials or talks. Keep your lesson within the scope of your lesson plan's goals. If you're deviating from your lesson plan's goals, you're probably covering too much material. Practice, practice, practice your talk/tutorial/lesson, and time it.

And consider whether your lesson plan is too ambitious.

Know what to hold back

- Answer the question asked
- Nuance isn't always helpful
- Teach the pool, not the ocean
- Don't cover too much material

Foster a love of learning

Reward curiosity

Children ask the best questions, until it's crushed out of them by tired adults

We all start out curious and full of wonder.
Children ask questions like "Why is the sky blue?" Questions that adults tend not to ask. Things that adults tend to take for granted.
But always: "Why?" and "What if ...?"
New learners start off asking these great questions too. Encourage this curiosity. Don't crush it out of them.

"I don't know, let's find out"

This is a magic phrase.

This magic phrase encourages questions, rewards curiosity, and is a reminder to us all that you don't need to know everything. It's a humbling phrase. It's an honest phrase. Learning never ends.

"Great question."

"I love your instincts!"

Here's another important phrase to remember.

Don't be annoyed when the learner is mentally jumping ahead a few slides and asking about material you're about to cover -- that's a sign that you're doing a good job! They're extrapolating from something that you've taught them.

Celebrate progress

Positive feedback encourages growth

Negativity stifles learning

Keep this in mind when posting on Stack Overflow, please, and again -- remember your journey. You didn't become an expert overnight.

You can be the mentor you always wished you had. You can be that person to someone else.

Move about

Teaching concepts through movement makes for unforgettable experiences

It will be rare that anyone will remember your slides. No matter how good they are or how much time you spent on them. People might remember your talk if they enjoyed it. But people will definitely remember activities they did as part of your lesson.

Move about

(A photo of people holding up cards that spell out "Python")

Here's a photo from lesson 1 with how we teach slicing - we have six people holding up cards, each with one character on them. We call out slicing numbers and people step forward if they're part of that slice.

We teach enumerate( ) similarly to how we teach for loops -- remember the rule of "use examples people already understand" -- once people have a strong grasp of for loops, we teach enumerate( ). First, we take attendance again just like a normal for loop; then we add the enumerate( ) in and ask attendees to stand up and say their name AND their position in the list.

To teach zip( ), we form two lines single-file and pair people off like they're walking in a wedding.

To teach while loops, I hold a handful of candy and my volunteer will ask "Hey Shannon, can I have a piece of candy?" until I'm all out.

Use props! Move about! These will make for memorable experiences.

We learn by teaching

Teaching reinforces what we've learned

Teaching also forces us to think flexibly, humbles us, and creates important bonds.
Be open to learning new techniques and perspectives from your students and other teachers!

Your students learn by teaching, too

Teaching reinforces what we've learned

Are there opportunities for students to teach others?

This concept is so important, it gets two slides. Because as a teacher, you should remain open to learning by teaching but also -- students will learn by teaching as well.

One way I love to do this in Hear Me Code is when we're installing Python, I like to walk one person through the installation, downloading the slides, and running their first "Hello world" then I ask them to walk their neighbor through it -- right away it reinforces what they just learned -- and creates an opportunity for people at the same table to connect with one another.

Foster a love of learning

- Reward curiosity
- "I don't know, let's find out"
- "Great question!"

Foster a love of learning

- Celebrate progress
- Move about
- We learn by teaching
- Your students learn by teaching too

You're an expert.

You know how to teach like one.

Say hi: shannonvturner.com

So now, you are an expert. You know how to teach like one, and thank you for attending.

I'd love to hear from you about how you're incorporating your new learnings into how you're teaching and mentoring others.

You're an expert.

You know how to teach like one.

Get the slides:
shannonvturner.com/pycon

As promised, here's where you can get the slides for this talk.

When the video for this talk is all recorded and processed, I'll post it here too.

Thank you everyone!