Python: From Apprentice to Master

Shantnu Tiwari

http://pythonforengineers.com/

So you want to get better at Python?

I have divided the type of programmer you can be into different stages. A thing to keep in mind is, these are stages I've just made up, and are obviously not as fine grained as they could be. I still think they are useful, as to parrot the cliche, "You don't know where you are going till you know where you are."

Yeah, sorry for that. If I repeat another cliche, I give you permission to slap me.

So without further ado, here are the stages of Python. My advice is to read through all the sections, even if you think you are at a higher level.

1. Beginners

1a. Those new to programming itself

This is the hardest stage to be, because the advice you will get here is usually crap. Most programmers (at least the ones who give advice online) started programming two hundred years ago, when humans still lived in caves, hunted dinosaurs for lunch and programmed big mainframes in Cobol using punch cards.

I myself started programming in the mid-90s, when I was about thirteen or fourteen years old. By the way, it's a bullshit myth that you need to start programming when you're a kid to become any good. My dad started programming at forty, and he is the one who inspired me.

Back when I started, there weren't that many choices to begin. I knew of only a few languages, tools were very basic, mainly command line compilers and a basic code editor. Sure, there were better tools, but they weren't free or even cheap.

You had to learn by a lot of trial and error, books were rare and expensive, and if you got stuck, there was no StackOverflow to ask questions to. You just worked through it, asked friends (if you were lucky to have friends who knew programming, but usually, people who knew programming didn't have friends).

The problem is, most books/tutorials nowadays still assume we are living in the dark ages. They recommend you start writing command line programs, struggling alone, wasting hours trying to get stuff working, trying a hundred different things, the sort of stuff we had to do in the dark ages.

Good stuff, if your time has no value.

And that's the problem with most advice to people starting out. It's given by people who learnt in a different environment, and have no idea what problems beginners face.

To Learn Python: I recommend something like Codecademy to pick up the basics. The problem with Codecademy is that it's entirely online. You need to download Python and get it running on your own machine.

After that, I recommend solving the first 10-15 Project Euler problems, but **no more**. They aren't that useful beyond understanding the basics.

1b. If already know one language

Use Codecademy to pick up the basics, and then try to write something you've written in our own language in Python.

Be careful not to write Java in Python. I've seen dozens of examples online which were

basically five hundred lines and three classes to print hello world. Now you might ask, how do you know you haven't ended up with Python Java? That's the next stage.

Check List for beginners:

* Do you understand the syntax of Python?

* If given a basic problem (ie, something that can be solved without using libraries), could you just go ahead and implement it? I'm talking about Project Euler type problems.

* Again, for basic problems, could you look at someone else's code and follow it (provided the code is cleanly written)?

Intermediate stage

A beginner knows the syntax of Python, can solve a few simple / mathematical problems. You know all these websites that recommend you solve hundreds of mathematical / Project Euler problems? That seems to be the general consensus in most forums. But if that's all you do, you'll never rise above a beginner.

I read this really funny comment on the Learn Python Reddit:

"I can print every bleepin' number from one to four million, but how does that help me learn Python?"

And that's the problem of always staying at beginner level.

The next stage

Knowing how to code is good enough if you are a beginner. But if you want to pass beyond the beginner, you need to write code that is clean and readable. Clean code means descriptive variable names. So if you are writing code to add two numbers, don't do:

```
# Add two numbers
t =  m1 + m2
```

While the above code has comments, they are useless, since the comment is just repeating the code. Instead, we could remove the comment by using better variable names:

```
total_marks = marks_english + marks_maths
```

If you want to write great code, the book I recommend, in fact, the *only* book I recommend, is: Code Complete by Steve McConnell. While I've read dozens of books, none comes close to its practicality and simple, clear advice.

One way to ensure that you write clean code is to follow a coding standard. Pep8 is the official Python standard, but I find that people follow it a bit too religiously. If you search for "Writing clean / good code Python" on Google, half the results will recommend following pep8, but here's the problem: They pretend it's the ONLY way, when it's just one way. This religiousness in Python's community really puts me off. Especially as many key Python libraries don't follow pep8: Unittest and PyQt are the two that come to mind.

Which is why I recommend, choose one standard. As long as you stay consistent, it doesn't matter what coding practice you use.

The only other tip I will give is: Don't over engineer. I read all these blogs where it's impossible to follow anything because the programmer has created five classes, and all they do is call one function that prints Hello World on the screen, and sometimes it doesn't even do that because there is a bug.

The problem is, many of these blogs are written to impress others, and the writer puts in every cool feature s/he read of. Or they think, "Woah! I better use classes, or someone will tell me off." And they add unnecessary complexity that makes it hard to follow the code. Grow some backbone, be proud of your work, don't let random strangers on the internet bully you around.

Follow KISS: Not the rock band, the principle. Keep It Simple, Stupid.

So write your code as simply as you can. If you find yourself repeating code, put it in a function. If you find yourself using a group of related functions a lot, put them in a class. If you

want to share your code with someone, without them having to dig through it, create a library.

But always go from simple to complex, not the other way round.

But that still isn't enough to cross the beginner's level. The intermediate stage is huge, I guess we could break it further, but I won't for now.

Most modern programming is done via libraries, and you need to have knowledge of these libraries to be productive. Here is a short list of useful topics to study:

1. Web app development, with a full stack framework like Django, or a minimal like Flask.
2. Scientific computing with Numpy/Scipy
3. Data analysis, using either pandas or numpy
4. Databases, and how to talk to them using Python. Both direct and ORM
5. Running command line programs from Python
6. Testing with Python
7. Gui apps
8. Any other you find interesting!

Choose 3-4 of the above topics. Any that interest you. Or that you find challenging, or you feel can help your career. Read as many blogs as you can on them. Run the examples, see that you can get the code running. You will face problems, don't give up. Look in Stackoverflow, other people's code, blogs etc. Learning how to search for help, struggling and finding solutions in unexpected places is one of the key factors to go from beginner to master.

Note I said 3-4 topics. You can look at all the topics, if you feel like it.

If you are short of time, just read blogs like a magazine, without trying anything. You will still learn stuff, and later on, if you want to use something, you'll at least have an idea how to approach the problem (or at least, how to find the blog you read!).

By looking at other peoples code, you'll learn how to write proper Python code. This will stop you from writing Java in Python.

Check List for intermediate programmers:

***** Have you read Code Complete? Do you understand the basics of writing good code?

* Do you understand that code is read more times than it is written, so your solution must not only be clean, but simple to follow as well (KISS principle)?

* Have you read a fair number of blogs, and/or looked at other people's code, so you know how good Python code is written?

* Do you understand 3-4 libraries, such that you could use them in your own code?

The Advanced Stage

This is where I will go ahead and disagree with all the advice out there (as if I have been model of agreement till now!)

Most advice online says: Start contributing to open source projects. The bigger and more glamorous(sexy!), the better.

There is only one problem with this: Most open source projects are too complex. It will take you weeks, if not months, to understand it all. The other problem is that most of the important decisions would have been made. You look at the code, it will not make sense, because someone made a decision six years ago and didn't consult you. The code will be like a black box: you won't be able to understand its innards much. All you will be able to do is fix minor bugs. If that's your game, then hey, go wild.

I won't even go into the politics of many of these projects. It can get real nasty, especially to newcomers, who don't know how the game is played. You can face a situation where you go, "Hey, am I being insulted for volunteering my time? £$%^ this."

So what do I recommend? Choose a real world project, something that interests you, and just bleepin' build it. But just one caveat: Choose something that you can finish in 2 or 3 months, max. Otherwise you'll never finish it and abandon it. Even better, choose something you can build in one month.

In parallel, look at places where beginners struggle. You will see they struggle with basic things like installing libraries, Python versions, 32vs64 bit errors. They get hate on most forums. Try to learn things like how to install libraries on both Windows and Linux (optionally Mac). Learn a bit about Linux servers, as Python is primarily a server language.

As for the project, choose something you can show to the world. So no mathematical problems, no print all Fibonacci numbers from four to four billion. Build something real you can show to your grandma or your third cousin you only meet at Christmas.

And the secret…

The most important thing: Document your work. Check it into Github, blog about it, share it with the world.

Here is the key difference between intermediate and advanced programmers: Advanced programmers know how to share and present their ideas. Knowing how to express yourself is important as being technically proficient. That's where most advice on Internet fails. It only focuses on technical parts. Contribute to open source, learn yet another library, keep beautifying your code etc etc etc. No one cares. About how pretty your code is. Seriously.

To be slightly vain and quote my own book **Stop Being the Hamster** :

"*You might be the best programmer in the world. But if all you do is hide in your cubicle, no one will know. You might be the best writer in the world, but if you never publish anything, no one will hear of you. You might be the best artist in the world, but if you hide your painting in your toilet, no one will care (except for those unlucky enough to get a bad tummy in your house).*

Most people spend years developing their skills but spend no time in learning how to market themselves. In the real world, how competent you are is only half the story. The other half is: how do other people view you? And not just your bosses, but your colleagues, the wider community you are a part of and the general population you serve.

Your skill is useless without visibility. That's the reason you don't get any respect at work. While you think you are doing a great job, others don't think so. The largest reason being they aren't aware of it (which itself is caused by them not caring about you. People only care about themselves, remember?)"

Find the things you hated in blogs

As an intermediate programmer, I asked you to read a lot of blogs. You must have found a hundred things you hated. Skipping steps, jumping over basics, forgetting to include background/prerequisite knowledge, over engineering, focusing only on fake toy world problems etc. Make a list of things that pissed you off, and don't repeat them. As the saying goes: Don't get mad, write a blog.

When you write your own blog, **don't make those mistakes**. There you go, my super duper secret formula to writing successful blogs. Yes, I accept tips. Thanks.

Hallmark of an advanced programmer

An advanced programmer is one who can take a fairly complex autonomous project and execute it. Can teach stuff to beginners. Most importantly, has patience, not attitude. You know all these online forums, where if you post a question, they ask you to Read The Fucking Manual(RTFM)? Well, I always go, you read the fucking manual. Even better, why don't you write the software so that it's easy to use? Why don't you put up an easy to use FAQ instead of a 600 page manual?

Don't be that guy, the rude asshole on forums insulting beginner programmers. If all the beginners struggle with the same thing, that means the system is broken. Find a better way to teach them, don't insult them.

I just know that most of you will ignore this advice. It's far easier to find some obscure open source project, write some code for it, and then become an "expert". And hey, that's fine. Just remember, you are not an expert till other people say you are. And if all you are doing is acting cool and smug, no will ask you for advice, and no one will think you are an expert.

To summarise:

Build something independently, something useful you can show to others. Something useful, not a toy mathematical problem. Something you can show to your family. Check the code in into Github or something similar. And then blog about what you did, and what you achieved. Don't worry about no one reading your blog. You aren't doing this to fulfil your ego. You are doing to show to the world, and more importantly, yourself, that you got the chops. You can do it.

And if you are scared of people laughing at you, remember this: Someone somewhere will always hate what you do. Ignore the haters.

And if you are looking for permission to do something great in your life, let me say so now: **I give you Permission to be Awesome**. Now go forth and build something.

Check List for advanced programmers:

* Can you start and complete autonomous projects? Autonomous means something you aren't following from a blog. Complete means you can ship it, show it to people, maybe even sell it. So if it's a webapp, that means running on a server. If it's a desktop app, it must have an installer, must be easy to use. In both cases, must have good documentation.

* Are you helping beginners? In forums, Stackoverflow, etc? Have you started seeing a pattern of places where beginners struggle?

* Do you have a blog that others can follow? It doesn't matter how many people read it. Do you have one? Are you documenting your work?

* **Most important**: Do you understand coding is more than banging out code like a monkey? That you have to collaborate with others, design software that is easy and intuitive to use, that the people part of coding (communication) is as important, if not more, than the technical part? If you read the books in the next section, you will be well on your way to doing so.

Books I recommend:

All these books are available from the library. If not, someone around you should have a copy. There is no need to spend a lot of money buying them,

1. **Code Complete** is a great book, which everyone should read. It's a great overview about what it takes to write good code.

2. **The Mythical Man Month** is a great book to know about software project management. It contains Brook's Law, which I am fond of quoting to managers:

Adding more people to a late project makes it later.

Or, in its more pedestrian version:

It takes nine months to make a baby, no matter how many women you put on the job.

You'll have to read the book to find out why, but it has to do with the fact that most time wasted in software companies is wasted on bad communication.

Another law Brooks gave was the *No Silver Bullet* law: That is, there is no one thing you can do that will suddenly make you a super programmer (or make your project successful). Instead, there are a hundred small things you must do.

The surprising thing is, the book was written way back in 1975. So everything programmers struggle with today: Stupid time budgets, ballooning and unstable code base, constantly late projects, failed projects, all these things were known back in the 70s. You'd think we'd have wised up by now. Maybe because more programmers (and their managers) haven't read this book.

3. **The Inmates are Running the Asylum**: Many programmers hate this book. And I mean really hate. But I love it. Alan Cooper, the author, makes the point that the reason software frustrates us is because it is rude. Constantly asking us, "*Are you sure?*" Always second guessing us. Giving priority to what the programmer wanted, not what the user wants.

This is the reason many non-technical people hate computers. Rather than taking offence, us programmers should accept that there is a human being on the other side, and we have to treat them with respect.

And this isn't just about being polite anymore. People will pay more money for software that is easy to use. Just ask Amy Hoy, who makes half a million dollars a year from time tracking software that a college student could make. Why? Because it is easy to use. People pay good money for easy to use software/hardware. Again, ask Apple.

Related to the above themes are two more books: **The Design of Everyday things** and **Don't make me think**.

I am not a designer, but I think that if more programmers knew about these things, we'd have much better software, and we would all be richer as a result.

And that's it.

Now start your advanced Python project!

Do contact me to let me know how you are doing. If you are stuck, or need ideas, or just want to share your feedback, contact me at:

http://pythonforengineers.com/contact/

If I love what you are doing, I may even feature you on my blog.

Looking forward to hearing from you!
Shantnu