Python progression path - From apprentice to guru

I've been learning, working, and playing with Python for a year and a half now. As a biologist slowly making the turn to bio-informatics, this language has been at the very core of all the major contributions I have made in the lab. I more or less fell in love with the way Python permits me to express beautiful solutions and also with the semantics of the language that allows such a natural flow from thoughts to workable code.

What I would like to know is your answer to a kind of question I have seldom seen in this or other forums. This question seems central to me for anyone on the path to Python improvement but who wonders what his next steps should be.

Let me sum up what I do NOT want to ask first ;)

- I don't want to know how to QUICKLY learn Python
- Nor do I want to find out the best way to get acquainted with the language
- Finally, I don't want to know a 'one trick that does it all' approach.

What I do want to know your opinion about, is:

What are the steps YOU would recommend to a Python journeyman, from apprenticeship to guru status (feel free to stop wherever your expertise dictates it), in order that one IMPROVES CONSTANTLY, becoming a better and better Python coder, one step at a time. Some of the people on SO almost seem worthy of worship for their Python prowess, please enlighten us :)

The kind of answers I would enjoy (but feel free to surprise the readership :P), is formatted more or less like this:

- Read this (eg: python tutorial), pay attention to that kind of details
- Code for so manytime/problems/lines of code
- Then, read this (eg: this or that book), but this time, pay attention to this
- Tackle a few real-life problems
- Then, proceed to reading Y.
- Be sure to grasp these concepts
- Code for X time
- Come back to such and such basics or move further to...
- (you get the point :)

I really care about knowing your opinion on what exactly one should pay attention to, at various stages, in order to progress CONSTANTLY (with due efforts, of course). If you come from a specific field of expertise, discuss the path you see as appropriate in this field.

EDIT: Thanks to your great input, I'm back on the Python improvement track! I really appreciate!

python

edited May 21 '13 at 18:38

community wiki
12 revs, 3 users 69%
Morlock

19 Answers

I thought the process of Python mastery went something like:

1. Discover list comprehensions
2. Discover generators
3. Incorporate map, reduce, filter, iter, range, xrange often into your code
4. Discover Decorators
5. Write recursive functions, a lot
6. Discover itertools and functools
7. Read Real World Haskell (read free online)
8. Rewrite all your old Python code with tons of higher order functions, recursion, and whatnot.
9. Annoy your cubicle mates every time they present you with a Python class. Claim it could be "better" implemented as a dictionary plus some functions. Embrace functional programming.
10. Rediscover the Strategy pattern and then all those things from imperative code you tried so hard to forget after Haskell.
11. Find a balance.

edited Jan 28 '13 at 10:28

community wiki
3 revs, 2 users 96%
wheaties

@wheaties Super nice list, love it! You seem to have a bias towards functional programing and away from object oriented... is there a particular reason? Has the Haskell experience rerouted you towards this approach? – Morlock Apr 5 '10 at 12:58

8 Yes and no. I find I write better and more concise code the closer I get to functional programming. That said, there are plenty of places where things are expressed more clearly in an imperative manner. There's a reason Real World Haskell is sold out at Python conventions, it makes you a better programmer. My advice, explore as many facets of Python as you can, then try a functional language. – wheaties Apr 5 '10 at 13:32

@wheaties gets the reply since it is both the closer to the format asked for and very inspiring. Thanks for taking your time to reply! – Morlock Apr 7 '10 at 1:05

8 @wheatis I'm now scared if I read into Haskell I'll abandon Python for good. – Humphrey Bogart Apr 10 '10 at 2:16

1 Beginners: Dictionary then later on... Mastery: Metaprogramming – gath Apr 16 '10 at 14:25

You better learn range/xrange before you learn listcomps... – djc Apr 19 '10 at 12:00

Could you elaborate on >>it could be "better" implemented as a dictionary plus some functions.<< ? – Koobz Sep 30 '10 at 15:33

@Koobz I'm trying to put into words exactly what I meant by that statement which after rereading this long time almost appears as if I'm promoting procedural programming. That is not the case. The comment stems from the way a Python class is organized and what hidden dangers lurk via the "**dict**" property. I really can't expand more in the small space of this comment. – wheaties Sep 30 '10 at 19:05

8 Can i heartily recommend Dive Into Python as a way of doing #1, 2, 3, and 4? That book helped me tons – Claudiu Oct 4 '10 at 17:21

Can I replace #7 with "Read SICP"? – Limbo Peng Nov 11 '10 at 7:22

@E.T replace it with SICP if you want, ML, Lisp, Clojure, Scala, APL or any other functional language. However, you'd be best served with one that is statically typed, seeing as Python is dynamically typed. – wheaties Nov 11 '10 at 12:36

The Strategy link has moved to wikibooks, here is the new link en.wikibooks.org/wiki/Computer_Science_Design_Patterns/… – rapadura Jun 12 '11 at 18:12

We can live pretty well without filter, map, reduce. Can't we? Guido wanted to cut them three from Python 3000 – Peter Long Jul 24 '11 at 12:43

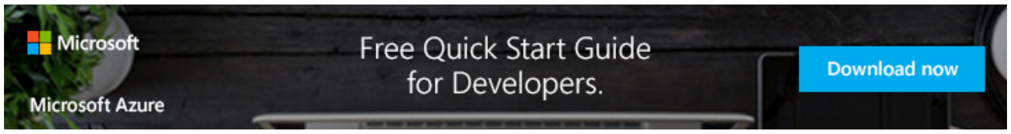
U can read RWH here : book.realworldhaskell.org/read – gregory561 Aug 22 '11 at 11:01

The Strategy-Link does no longer contain the content. Here is the content: en.wikipedia.org/w/… – Martin Thoma Jun 25 '12 at 5:40

Loved it, but have 2 comments. 1 filter, map, reduce mostly go away with good understanding of comprehension. Recursion is good for understanding, but bad in practice for a language like python. It should be avoided in production code. – cmd Mar 21 '13 at 16:11

@cmd in Python 2.x map is pushed down into the bowls of C while list comprehensions remain in Python. see: stackoverflow.com/questions/1247486/… – wheaties Mar 22 '13 at 18:31

After all you have done. This is more interesting "Find a balance". True. – Akshay Patil Jul 25 '13 at 4:59



One good way to further your Python knowledge is to **dig into the source code of the libraries, platforms, and frameworks you use already.**

For example if you're building a site on Django, many questions that might stump you can be answered by looking at how Django implements the feature in question.

This way you'll continue to **pick up new idioms, coding styles, and Python tricks**. (Some will be good and some will be bad.)

And when you see something Pythony that you don't understand in the source, **hop over to the #python IRC channel** and you'll find plenty of "language lawyers" happy to explain.

An accumulation of these little clarifications over years leads to a much deeper understanding of the language and all of its ins and outs.

answered Apr 4 '10 at 1:16

community wiki
dkamins

18 +1 for "read other people's code, frequently" – FogleBird Apr 4 '10 at 1:19

@dkamins Thanks for the advices. Sure will dig into specific libraries (for me, that would be Biopython). I wasn't aware of the #python IRC channel. Will also give it a try. :) – Morlock Apr 4 '10 at 3:57

1 I would add to this that instead of simply installing a django extension/plugin, try adding the code manually using a git branch. This will force you to actually *read the code* you are adding to the project. – g33kz0r Apr 11 '10 at 3:21

14 Am curious why **use IRC** when the rest of us could benefit from the Question? – Tshepang Dec 1 '10 at 16:38

1 +1 for "idioms, styles and tricks". Living a real CULTURE made by real people in contact (even if this contact mean mostly reading each other's code in this case) is what has ever made human knowledge EVOLVE over time. – heltonbiker Jul 17 '13 at 22:51

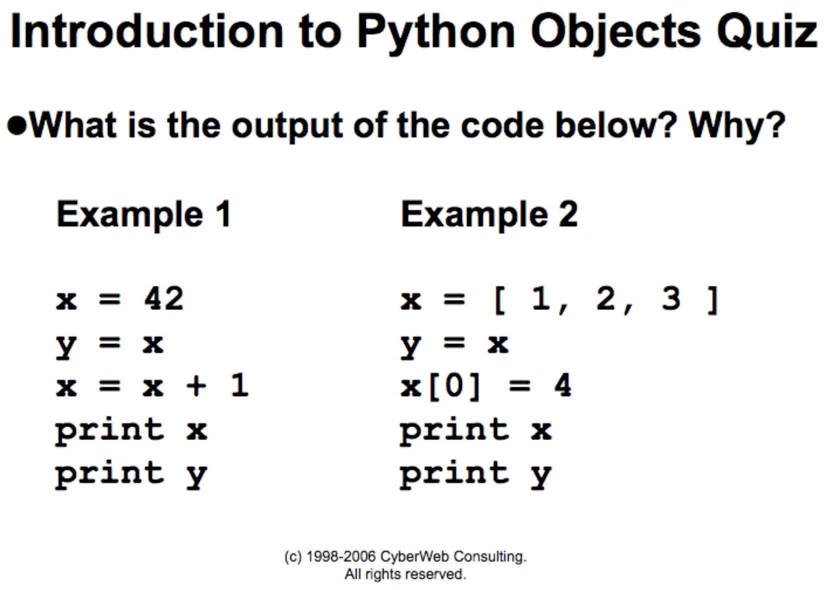
Understand (more deeply) Python's data types and their roles with regards to memory mgmt

As some of you in the community are aware, I teach Python courses, the most popular ones being the comprehensive Intro+Intermediate course as well as an "advanced" course which introduces a variety of areas of application development.

Quite often, I get asked a question quite similar to, "Should I take your intro or advanced course? I've already been programming Python for 1-2 years, and I think the intro one is too simple for me so I'd like to jump straight to the advanced... which course would *you* recommend?"

To answer their question, I probe to see how strong they are in this area -- not that it's really the best way to measure whether they're ready for any advanced course, but to see how well their basic knowledge is of Python's objects and memory model, which is a cause of *many* Python bugs written by those who are not only beginners but those who have gone beyond that.

To do this, I point them at this simple 2-part quiz question:


Introduction to Python Objects Quiz

- What is the output of the code below? Why?

Example 1

```
x = 42
y = x
x = x + 1
print x
print y
```

Example 2

```
x = [ 1, 2, 3 ]
y = x
x[0] = 4
print x
print y
```

(c) 1998-2006 CyberWeb Consulting.
All rights reserved.

Many times, they are able to get the output, but the *why* is more difficult and much more important of an response... I would weigh the output as 20% of the answer while the "why" gets 80% credit. If they can't get the why, regardless how Python experience they have, I will always steer people to the comprehensive intro+intermediate course because I spend one lecture on objects and memory management to the point where you should be able to answer with the output and the why with sufficient confidence. (Just because you know Python's syntax after 1-2 years doesn't make you ready to move beyond a "beginner" label until you have a much better understanding as far as how Python works under the covers.)

A succeeding inquiry requiring a similar answer is even tougher, e.g.,

Example 3

```
x = ['foo', [1,2,3], 10.4]
y = list(x) # or x[:]
y[0] = 'fooooooo'
y[1][0] = 4
print x
print y
```

The next topics I recommend are to understanding reference counting well, learning what "interning" means (but not necessarily using it), learning about shallow and deep copies (as in Example 3 above), and finally, the interrelationships between the various types and constructs in the language, i.e. lists vs. tuples, dicts vs. sets, list comprehensions vs. generator expressions, iterators vs. generators, etc.; however all those other suggestions are another post for another time. Hope this helps in the meantime! :-)

ps. I agree with the other responses for getting more intimate with introspection as well as studying other projects' source code and add a strong "+1" to both suggestions!

pps. Great question BTW. I wish I was smart enough in the beginning to have asked something like this, but that was a long time ago, and now I'm trying to help others with my many years of full-time Python programming!!

edited Oct 4 '10 at 17:19

community wiki
3 revs, 2 users 99%
wescpy

Wesley Chun! Morlock, this is the kind of guy you were talking about in the bolded part of your question. A deeply knowledgable guy who also a very gifted teacher, I got a huge boost from the book. – unmounted Apr 5 '10 at 19:11

thx @bvmou! i could only answer this question because i had to go through this learning process like everyone else... only mine was 13 years ago! :-) thx for plugging corepython.com tho! :-) – wescpy Apr 9 '10 at 4:53

@wescpy Can you explain me the output for example3 or point me to a good resource to learn this stuff.Thanks. – Sankalp Aug 13 '13 at 6:17

1 The trick is that only object references are copied, not the objects themselves. That's all there is to it... hope it helps! I gave a talk on this a few weeks ago actually, at OSCON. I was pleasantly surprised when someone told me it was the highest rated Python talk at the conference! You can download the slides here: oscon.com/oscon2013/public/schedule/detail/29374 – wescpy Aug 14 '13 at 4:20

The output is SyntaxError: invalid syntax, isn't it? Why? Because I have used Python 3, and `print` isn't a statement. – xfix Oct 1 '13 at 14:07

Check out Peter Norvig's essay on becoming a master programmer in 10 years: http://norvig.com/21-days.html. I'd wager it holds true for any language.

answered Apr 4 '10 at 21:14 community wiki twneale

2 Right, I had come across this one some time ago, but didn't read through. Thx! – Morlock Apr 4 '10 at 22:02

Understand Introspection

- write a `dir()` equivalent
- write a `type()` equivalent
- figure out how to "monkey-patch"
- use the `dis` module to see how various language constructs work

Doing these things will

- give you some good theoretical knowledge about how python is implemented
- give you some good practical experience in lower-level programming
- give you a good intuitive feel for python data structures

edited Sep 23 '11 at 17:45 community wiki 3 revs, 3 users 85% Mark Harrison

1 "Write a `type()` equivalent" that would be very hard if you take the full metaclassy spec of `type`! – asmeurer Jun 4 '13 at 2:31

```
def apprentice():
  read(diveintopython)
  experiment(interpreter)
  read(python_tutorial)
  experiment(interpreter, modules/files)
  watch(pycon)

def master():
  refer(python-essential-reference)
  refer(PEPs/language reference)
  experiment()
  read(good_python_code) # Eg. twisted, other libraries
  write(basic_library)   # reinvent wheel and compare to existing wheels
  if have_interesting_ideas:
     give_talk(pycon)

def guru():
  pass # Not qualified to comment. Fix the GIL perhaps?
```

answered Nov 12 '10 at 6:22 community wiki amit

1 Hehe :) Fun format! Those are nice ideas, and comforting, since I already have done most of it (except the pycon parts). Some guru care to add the definition to the 'guru()' function? Thanks! – Morlock Nov 12 '10 at 12:45

4 guru() should be pass. nothing can describe guru, let it pass – inv May 16 '11 at 8:22

1 If you fix the GIL then I think that puts you at a level *HIGHER* than guru, but just below benevolent dictator. ;) – Adam Parkin Mar 1 '12 at 12:36

I'll give you the simplest and most effective piece of advice I think anybody could give you: **code**.

You can only be better at using a language (which implies understanding it) by *coding*. You have to actively enjoy coding, be inspired, ask questions, and find answers by yourself.

Got a an hour to spare? Write code that will reverse a string, and find out the most optimum solution. A free evening? Why not try some web-scraping. Read other peoples code. See how they do things. Ask yourself what you would do.

When I'm bored at my computer, I open my IDE and code-storm. I jot down ideas that sound interesting, and challenging. An URL shortener? Sure, I can do that. Oh, I learnt how to convert numbers from one base to another as a side effect!

This is valid whatever your skill level. *You never stop learning.* By actively coding in your spare time you will, with little additional effort, come to understand the language, and ultimately, become a guru. You will build up knowledge and reusable code and memorise idioms.

answered Apr 4 '10 at 23:14 — community wiki Humphrey Bogart

If you're in and using python for science (which it seems you are) part of that will be learning and understanding scientific libraries, for me these would be

- numpy
- scipy
- matplotlib
- mayavi/mlab
- chaco
- Cython

knowing how to use the right libraries and vectorize your code is essential for scientific computing.

I wanted to add that, handling large numeric datasets in common pythonic ways(object oriented approaches, lists, iterators) can be extremely inefficient. In scientific computing, it can be necessary to structure your code in ways that differ drastically from how most conventional python coders approach data.

edited May 25 '12 at 16:31 — community wiki 3 revs, 2 users 78% user503635

Thanks. It is certainly worth taking time to learn numpy and scipy to gain efficiency in memory use for large datasets! – Morlock Nov 11 '10 at 17:21

1 Probably add pandas and IPython to the list. – Eric Wilson Jul 18 '13 at 12:12

Google just recently released an online Python class ("class" as in "a course of study").

http://code.google.com/edu/languages/google-python-class/

I know this doesn't answer your full question, but I think it's a great place to start!

answered Apr 4 '10 at 0:32 — community wiki dkamins

Thanks @dkamins :) It looks more like a basic tutorial though. Any personal thoughts about a path to continual Python improvement? – Morlock Apr 4 '10 at 0:34

6 actually, these video lectures from google are geared more towards itermediate to advanced I think. code.google.com/intl/fi-FI/edu/languages/… – Tom Willis Apr 4 '10 at 18:49

Download Twisted and look at the source code. They employ some pretty advanced techniques.

answered Apr 4 '10 at 1:38 — community wiki Jason Christa

1 Nice. I dug out the twisted code from my python lib folder and will go through it. – Morlock Apr 4 '10 at 4:05

Thoroughly Understand All Data Types and Structures

For every type and structure, write a series of demo programs that exercise every aspect of the type or data structure. If you do this, it might be worthwhile to blog notes on each one... it might be useful to lots of people!

answered Apr 4 '10 at 4:08 community wiki Mark Harrison

Wow, I like that idea. This would surely strengthen one's (or my) knowledge of Python's fundamentals. – Morlock Apr 4 '10 at 13:50

This would not only be useful in Python, but also as in an abstract sense. – Humphrey Bogart Apr 4 '10 at 23:08

1 This is something I do as well, whenever I find a new construct I write a toy program showing how to use it. And put it online in a github repo: github.com/pzelnip/MiscPython – Adam Parkin Mar 1 '12 at 12:41

I learned python first by myself over a summer just by doing the tutorial on the python site (sadly, I don't seem to be able to find that anymore, so I can't post a link).

Later, python was taught to me in one of my first year courses at university. In the summer that followed, I practiced with PythonChallenge and with problems from Google Code Jam. Solving these problems help from an algorithmic perspective as well as from the perspective of learning what Python can do as well as how to manipulate it to get the fullest out of python.

For similar reasons, I have heard that code golf works as well, but i have never tried it for myself.

answered Apr 4 '10 at 2:32 community wiki inspectorG4dget

3 You mean this tutorial? Python 2: docs.python.org/tutorial/index.html; Python 3: docs.python.org/py3k/tutorial/index.html. – Humphrey Bogart Apr 4 '10 at 23:06

Thanks for posting the tutorial. I don't know why I haven't been able to find it. I must be selectively blind :O – inspectorG4dget Apr 5 '10 at 0:35

Learning algorithms/maths/file IO/Pythonic optimisation

This won't get you guru-hood but to start out, try working through the Project Euler problems The first 50 or so shouldn't tax you if you have decent high-school mathematics and know how to Google. When you solve one you get into the forum where you can look through other people's solutions which will teach you even more. Be decent though and don't post up your solutions as the idea is to encourage people to work it out for themselves.

Forcing yourself to work in Python will be unforgiving if you use brute-force algorithms. This will teach you how to lay out large datasets in memory and access them efficiently with the fast language features such as dictionaries.

From doing this myself I learnt:

- File IO
- Algorithms and techniques such as Dynamic Programming
- Python data layout
  - Dictionaries/hashmaps
  - Lists
  - Tuples
  - Various combinations thereof, e.g. dictionaries to lists of tuples
- Generators
- Recursive functions
- Developing Python libraries
  - Filesystem layout
  - Reloading them during an interpreter session

And also very importantly

- When to give up and use C or C++!

All of this should be relevant to Bioinformatics

Admittedly I didn't learn about the OOP features of Python from that experience.

answered May 18 '11 at 11:15 community wiki
Captain Lepton

3 I like the PE site, but let's be honest: it's a math site, not a programming site. Most of the problems (or at least most of the ones I've done) really boil down to knowing a few math tricks than anything programming language related. Oftentimes the good solutions in the forums get drowned out by the waves of naive solutions, etc. – Adam Parkin Mar 1 '12 at 12:46

On the contrary I found that it involved application of both programming and Maths skills and most importantly the *combination of the two* to solve real problems. You can't use PE to learn Python programming if you have poor Maths skills but you are only going to learn how to use a language by using it so it is ideal if your Maths is good enough. – Captain Lepton Mar 7 '12 at 9:23

Have you seen the book "Bioinformatics Programming using Python"? Looks like you're an exact member of its focus group.

answered Apr 4 '10 at 6:39 community wiki
Tim Pietzcker

2 After going rapidly through the book some time ago, I found that this book is not very interesting. It is geared towards very novice Python users and the problems themselves are not extremely interesting. Too bad, this is clearly an area where Python use is increasing. A more stimulating book would have been great. – Morlock Dec 14 '10 at 16:52

You already have a lot of reading material, but if you can handle more, I recommend you learn about the evolution of python by reading the Python Enhancement Proposals, especially the "Finished" PEPs and the "Deferred, Abandoned, Withdrawn, and Rejected" PEPs.

By seeing how the language has changed, the decisions that were made and their rationales, you will absorb the philosophy of Python and understand how "idiomatic Python" comes about.

http://www.python.org/dev/peps/

answered Apr 10 '10 at 0:40 community wiki
Greg Ball

Attempt http://challenge.greplin.com/ using Python

answered Dec 4 '11 at 11:50 community wiki
Nam Ngo

Teaching to someone else who is starting to learn Python is always a great way to get your ideas clear and sometimes, I usually get a lot of neat questions from students that have me to re-think conceptual things about Python.

answered Nov 12 '10 at 22:25 community wiki
Marc-Olivier Titeux

Indeed. I have already made 3 introductory courses to Python in the last 2 years. The crowd of pythoneers is steadily growing around me, and that's nice :) – Morlock Dec 14 '10 at 16:53

Not precisely what you're asking for, but I think it's good advice.

Learn another language, doesn't matter too much which. Each language has it's own ideas and conventions that you can learn from. Learn about the differences in the languages and more importantly `why` they're different. Try a purely functional language like Haskell and see some of the benefits (and challenges) of functions free of side-effects. See how you can apply some of the things you learn from other languages to Python.

answered Apr 4 '10 at 1:13

community wiki
Davy8

1 Haskell is intriguing, and I am drawn to Lisp, but I feel there is much for me still to discover within the Python world before I wish to move on. – Morlock Apr 4 '10 at 4:07

2 landoflisp (.com) has recently drawn my attention. – Jiaaro Nov 10 '10 at 20:03

If anyone's thinking about Lisp, I recommend Scheme – Tharindu Rusira Sep 20 '13 at 11:58

I recommend starting with something that forces you to explore the expressive power of the syntax. Python allows many different ways of writing the same functionality, but there is often a single most elegant and fastest approach. If you're used to the idioms of other languages, you might never otherwise find or accept these better ways. I spent a weekend trudging through the first 20 or so Project Euler problems and made a simple webapp with Django on Google App Engine. This will only take you from apprentice to novice, maybe, but you can then continue to making somewhat more advanced webapps and solve more advanced Project Euler problems. After a few months I went back and solved the first 20 PE problems from scratch in an hour instead of a weekend.

edited Sep 3 '13 at 5:46

community wiki
2 revs, 2 users 67%
hus787