Get started Open in app

244 Followers About Follow

5 Python Tricks That Made Me A Good Python Developer

Varun Singh Aug 31 · 6 min read

Source — Canva | Created By Varun Singh

Software developers create bugs on a daily basis. It's not completely their fault but it's good to understand these mistakes and constantly improve. Recently I listed top 3 mistakes that every software developer should avoid — You can read it here — Top 3 Software Development Mistakes You Should Avoid.

Three years back when I started as a Python developer, I was writing code by learning from the existing codebase. I was never much of a reader myself. This delayed my

Get started Open in app

Past 6 months, I have invested in reading and practicing Python more than ever. I came across few python tricks that I should have known right from the beginning. I am going to share these tricks in this article.

⚡1 — Use Lambdas Over Functions

Most of the time a code reviewer🧐 use to complain about my code is when I always used a python function that is getting called only once in its lifetime. The conversation with the reviewer got awkward😢 quickly when the function had a single line in it.

Both are the reasons to consider using **lambdas** instead of python functions here. The **lambda** keyword in Python provides a shortcut for declaring an anonymous function.

Lambda functions behave just like regular functions declared with the def keyword. They can be used wherever function objects are required. For example, this is how you'd define a simple lambda function carrying out an addition:

```
>>> add = lambda x, y: x + y
>>> add(5, 3) 8
```

Key takeaways from this trick are that whenever you end up writing a single line function that is getting invoked only once — consider using lambda functions instead. This will be more pythonic.

⚡2- Don't Underestimate The Decorators

I have always been scared of weird symbols my entire life. One of them was @ 😢. Let's say your manager comes to you and asks to implement an entry logging line for all the 20 APIs in the project server code.

And, since the manager is planning to deliver this to production in the next 2 days, you will either end up updating each API's code with duplicate logging lines and even end up breaking one of those API.

Well, that is a pickle you really don't want to be in. Instead, you can type the code for a generic @ **print_log** decorator and paste it in front of each API definition. For example,

Get started Open in app

```
(name)s - %(levelname)s - %(message)s')

def print_log(func):
    def log_decorator_wrapper():
        """log function begining"""
        logging.warning("[UserLogin]: API Called")

        return log_decorator_wrapper

@print_log
@login("/login", method=["POST"])
def user_login():
    ---- Some API Logic ---
```

Then you'll commit your code and can have a cup of tea(Maybe try Assam tea😋) and chill.

⚡3- Be More Pythonic When Writing Loops

Iterating over a sequence of structured or unstructured data is very common in any language. Another fact I would like to present here is most of us have never studied python and its internal ways very well while in our academic studies.

One thing to spot about a developer with a background in Java or other programming is how they write loops in python. For example, this is what I used to write when I first started with python -

```
my_list = ['Tom', 'Jerry']

i = 0
while i < len(my_list):
    print(my_list[i])
    i += 1
```

Now, this code snippet looks so simple, but here is one problem that your reviewer might comment(if he/she is a huge python geek 🤭) -

> ***This does not seems pythonic — The Reviewer***

Let me explain why. First, the code above keeps track of an index variable **i** and then increments it after the while loop.

Get started Open in app

```
for i, item in enumerate(my_list):
    print("{}: {}".format(i, item)
```

Python has inbuilt ways to keep track of the index while iterating over a sequence(in our case, a list) by using **enumerate()**. It will take care of tracking the index variable **i** and will automatically increment it.

This way it looks more pythonic than before and now your code reviewer will be happy to see this and you will be able to push your code to the master branch. And maybe, this time try Nilgiri tea ☕ and chill.

⚡4- Stop Using Old String Formatting Ways

When I started writing code in python, string formatting with percentage symbols was still a thing. And as developers, we learn more from existing code than anywhere else.

So I adapted using percentage symbols to create string formatting lines in my code. Now, let's talk what was the problem with this by looking at some example code below -

```
>>> 'Hey %s' % (name)
'Hey Varun'
```

Well, believe it or not, this style requires way more typing than ever. Python has de-emphasized using this style. It has not been deprecated in the latest versions but there are better ways to format strings now. For example -

```
>>> 'Hey {}'.format(name)
Hey Varun
```

Or

Get started Open in app

The above example shows how to print a string stored in a variable ***name***. These are much easier to remember and also require fewer symbols to be present while formatting a string.

⚡5- Use VirtualEnv More

Installing Python packages globally can also incur a security risk. Modifying the global environment often requires you to run the pip install command with superuser (root/admin) credentials.

More often we as developers are testing with newer packages or libraries while working with python. But making the mistake to install these dependencies directly in your project environment is not a good idea. Here is why -

Firstly, we all are humans and we may forget to uninstall in case we are about to ship the project to the testing team or a client with a quick update or maybe you don't need that dependency for a time being.

Secondly, if you are not using any environment or tools like docker, it will be very hard to track why the project setup is taking time, because the next time you run your code on a new server, and you forget to remove the import statement of that dependency, you may end up breaking the server code.

That is scary, right? Instead, you should use **Virtual Environments** when you are developing with new libraries or dependencies. This will keep your development environment isolated from the actual project environment.

You can create a virtual environment using **venv** in python by running a simple command as below -

```
$ python3 -m venv ./venv
```

Once the environment is created, you can enable this new environment by running this command -

Get started Open in app

Now you can easily install anything in this environment as this creates isolation with your global dependencies. You can read more about virtual env here — Create Python Virtual Environment In 2 Minutes

. . .

More About The Author😄

I am a full-time software engineer with 4+ years of experience in Python, AWS, and many more technologies. I have started writing recently as it helps me to read more. I am looking forward to sharing technical knowledge and my life experiences with people out there.

Register Here for my Programmer Weekly newsletter where I share interesting tutorials, python libraries, and tools, etc.

Github | LinkedIn | Twitter | Facebook | Quora | Programmer Weekly

. . .

Originally published at https://blog.varunsingh.in.

Programmer Weekly Newsletter

Subscribe to my programmer weekly newsletter for free where I will send you curated articles and tutorials straight to your inbox. Note: You can unsubscribe anytime.

Your email

Get started Open in app

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Python Technology Software Development Programming Machine Learning

Medium

About Write Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play