

# Writing Comments in Python (Guide)

by [Jaya Zhané](#) 📅 Nov 05, 2018 📖 18m 💬 37 Comments 🏷️ [basics](#) [best-practices](#)

Mark as Completed



🔗 Share

## Table of Contents

- [Why Commenting Your Code Is So Important](#)
  - [When Reading Your Own Code](#)
  - [When Others Are Reading Your Code](#)
- [How to Write Comments in Python](#)
  - [Python Commenting Basics](#)
  - [Python Multiline Comments](#)
  - [Python Commenting Shortcuts](#)
- [Python Commenting Best Practices](#)
  - [When Writing Code for Yourself](#)
  - [When Writing Code for Others](#)
- [Python Commenting Worst Practices](#)
  - [Avoid: W.E.T. Comments](#)
  - [Avoid: Smelly Comments](#)
  - [Avoid: Rude Comments](#)
- [How to Practice Commenting](#)
- [Conclusion](#)

[Watch Now](#)

This tutorial has a related video course created by the Real Python team. Watch it together with the written tutorial to deepen your understanding: [Writing Comments in Python](#)

When writing code in Python, it's important to make sure that **your code can be easily understood by others**. Giving [variables](#) obvious names, [defining explicit functions](#), and organizing your code are all great ways to do this.

Another awesome and easy way to increase the readability of your code is by using **comments**!

💬 Help

In this tutorial, you’ll cover some of the basics of writing comments in Python. You’ll learn how to write comments that are clean and concise, and when you might not need to write any comments at all.

### You’ll also learn:

- Why it’s so important to comment your code
- Best practices for writing comments in Python
- Types of comments you might want to avoid
- How to practice writing cleaner comments

**Free Bonus: 5 Thoughts On Python Mastery**, a free course for Python developers that shows you the roadmap and the mindset you’ll need to take your Python skills to the next level.

## Why Commenting Your Code Is So Important

Comments are an integral part of any program. They can come in the form of module-level docstrings, or even inline explanations that help shed light on a complex function.

Before diving into the different types of comments, let’s take a closer look at why commenting your code is so important.

Consider the following two scenarios in which a programmer decided not to comment their code.

### When Reading Your Own Code

Client A wants a last-minute deployment for their web service. You’re already on a tight deadline, so you decide to just make it work. All that “extra” stuff—documentation, proper commenting, and so forth—you’ll add that later.

The deadline comes, and you deploy the service, right on time. *Whew!*

You make a mental note to go back and update the comments, but before you can put it on your to-do list, your boss comes over with a new project that you need to get started on immediately. Within a few days, you’ve completely forgotten that you were supposed to go back and properly comment the code you wrote for Client A.

Fast forward six months, and Client A needs a patch built for that same service to comply with some new requirements. It’s your job to maintain it, since you were the one who built it in the first place. You open up your text editor and...

What did you even *write*?!

You spend hours parsing through your old code, but you’re completely lost in the mess. You were in such a rush at the time that you didn’t name your variables properly or even set your functions up in the proper control flow. Worst of all, you don’t have any comments in the script to tell you what’s what!

Developers forget what their own code does all the time, especially if it was written a long time ago or under a lot of pressure. When a deadline is fast approaching, and hours in front of the computer have led to bloodshot eyes and cramped hands, that pressure can be reflected in the form of code that is messier than usual.

Once the project is submitted, many developers are simply too tired to go back and comment their code. When it’s time to revisit it later down the line, they can spend hours trying to parse through what they wrote.

Writing comments as you go is a great way to prevent the above scenario from happening. Be nice to Future You!

### When Others Are Reading Your Code

Imagine you’re the only developer working on [a small Django project](#). You understand your own code pretty well, so you don’t tend to use comments or any other sort of documentation, and you like it that way. Comments take time to write and maintain, and you just don’t see the point.

The only problem is, by the end of the year your “small Django project” has turned into a “20,000 lines of code” project, and your supervisor is bringing on additional developers to help maintain it.

The new devs work hard to quickly get up to speed, but within the first few days of working together, you’ve realized that they’re having some trouble. You used some quirky variable names and wrote with super terse syntax. The new hires spend a lot of time stepping through your code line by line, trying to figure out how it all works. It takes a few days before they can even help you maintain it!

Using comments throughout your code can help other developers in situations like this one. Comments help other devs skim through your code and gain an understanding of how it all works very quickly. You can help ensure a smooth transition by choosing to comment your code from the outset of a project.

## How to Write Comments in Python

Now that you understand why it’s so important to comment your code, let’s go over some basics so you know how to do it properly.

### Python Commenting Basics

Comments are for developers. They describe parts of the code where necessary to facilitate the understanding of programmers, including yourself.

To write a comment in Python, simply put the hash mark # before your desired comment:

Python

```
# This is a comment
```

Python ignores everything after the hash mark and up to the end of the line. You can insert them anywhere in your code, even inline with other code:

Python

```
print("This will run.") # This won't run
```

When you run the above code, you will only see the output `This will run`. Everything else is ignored.

Comments should be short, sweet, and to the point. While [PEP 8](#) advises keeping code at 79 characters or fewer per line, it suggests a max of 72 characters for inline comments and docstrings. If your comment is approaching or exceeding that length, then you’ll want to spread it out over multiple lines.

### Python Multiline Comments

Unfortunately, Python doesn’t have a way to write multiline comments as you can in languages such as [C](#), [Java](#), and [Go](#):

Python

```
# So you can't  
just do this  
in python
```

In the above example, the first line will be ignored by the program, but the other lines will raise a [Syntax Error](#).

In contrast, a language like Java will allow you to spread a comment out over multiple lines quite easily:

Java

```
/* You can easily  
write multiline  
comments in Java */
```

Everything between `/*` and `*/` is ignored by the program.

While Python doesn’t have native multiline commenting functionality, you can create multiline comments in Python. There are two simple ways to do so.

The first way is simply by pressing the return key after each line, adding a new hash mark and continuing your comment from there:

Python

```
def multiline_example():
    # This is a pretty good example
    # of how you can spread comments
    # over multiple lines in Python
```

Each line that starts with a hash mark will be ignored by the program.

Another thing you can do is use multiline strings by wrapping your comment inside a set of triple quotes:

Python

```
"""
If I really hate pressing `enter` and
typing all those hash marks, I could
just do this instead
"""
```

This is like multiline comments in Java, where everything enclosed in the triple quotes will function as a comment.

While this gives you the multiline functionality, this isn't technically a comment. It's a [string](#) that's not assigned to any variable, so it's not called or referenced by your program. Still, since it'll be ignored at runtime and won't appear in the bytecode, it can effectively act as a comment. (You can [take a look at this article](#) for proof that these strings won't show up in the bytecode.)

However, be careful where you place these multiline "comments." Depending on where they sit in your program, they could turn into [docstrings](#), which are pieces of documentation that are associated with a function or method. If you slip one of these bad boys right after a function definition, then what you intended to be a comment will become associated with that object.

Be careful where you use these, and when in doubt, just put a hash mark on each subsequent line. If you're interested in learning more about docstrings and how to associate them with modules, classes, and the like, check out our tutorial on [Documenting Python Code](#).

## Python Commenting Shortcuts

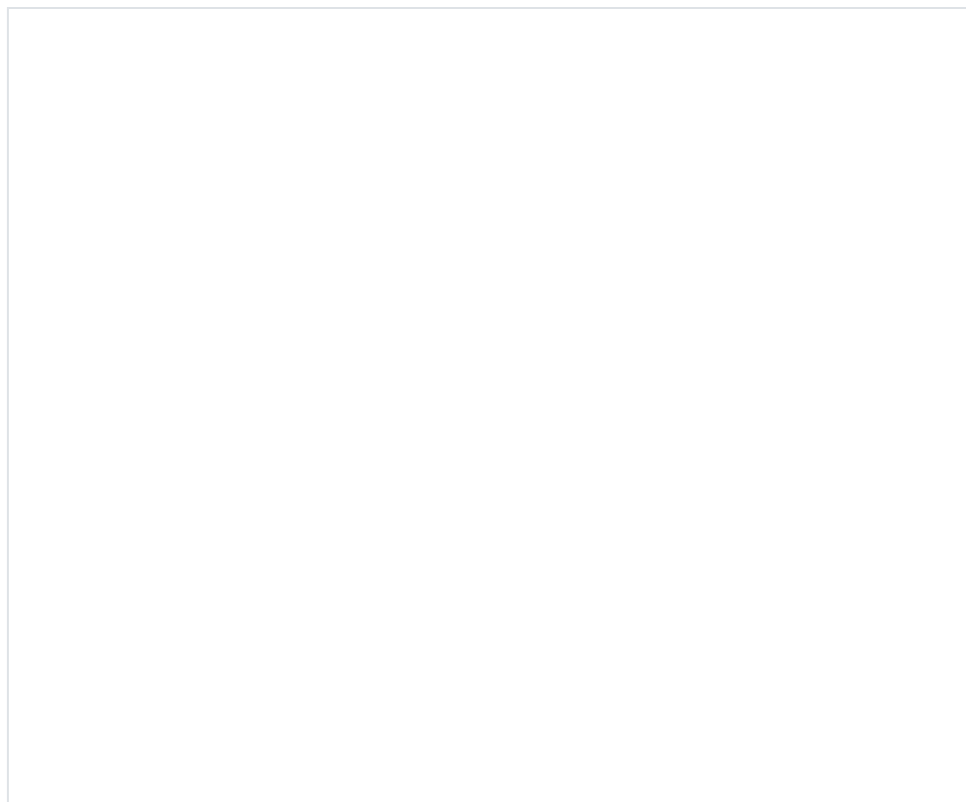
It can be tedious to type out all those hash marks every time you need to add a comment. So what can you do to speed things up a bit? Here are a few tricks to help you out when commenting.

One of the first things you can do is use multiple cursors. That's exactly what it sounds like: placing more than one cursor on your screen to accomplish a task. Simply hold down the `^ Ctrl` or `⌘ Cmd` key while you left-click, and you should see the blinking lines on your screen:



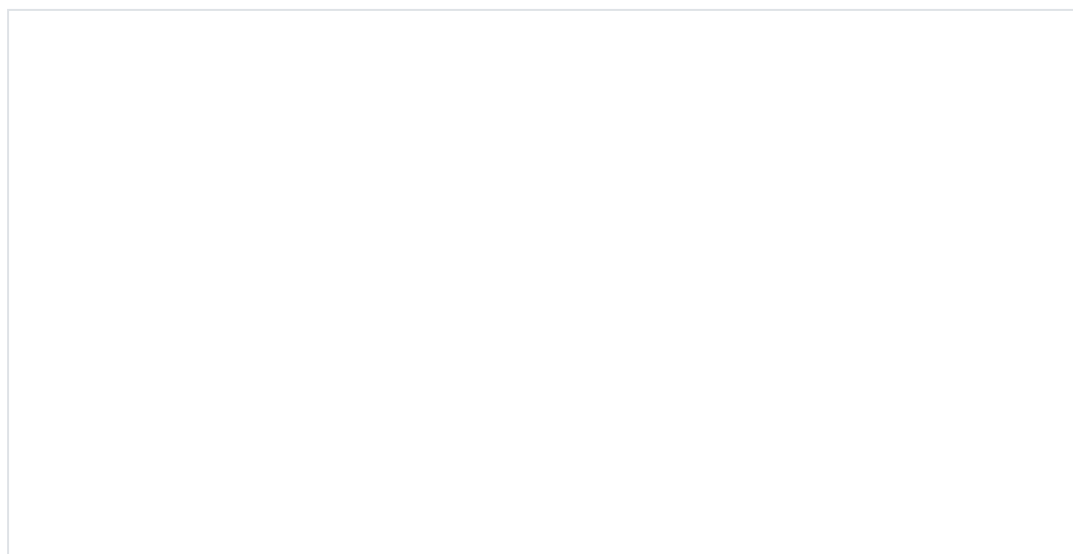
This is most effective when you need to comment the same thing in several places.

What if you've got a long stretch of text that needs to be commented out? Say you don't want a defined function to run in order to check for a bug. Clicking each and every line to comment it out could take a lot of time! In these cases, you'll want to toggle comments instead. Simply select the desired code and press `^ Ctrl` + `/` on PC, or `⌘ Cmd` + `/` on Mac:



All the highlighted text will be prepended with a hash mark and ignored by the program.

If your comments are getting too unwieldy, or the comments in a script you're reading are really long, then your text editor may give you the option to collapse them using the small down arrow on the left-hand side:



Simply click the arrow to hide the comments. This works best with long comments spread out over multiple lines, or docstrings that take up most of the start of a program.

Combining these tips will make commenting your code quick, easy, and painless!

## Python Commenting Best Practices

While it's good to know how to write comments in Python, it's just as vital to make sure that your comments are readable and easy to understand.

Take a look at these tips to help you write comments that really support your code.

### When Writing Code for Yourself

You can make life easier for yourself by commenting your own code properly. Even if no one else will ever see it, *you'll* see it, and that's enough reason to make it right. You're a developer after all, so your code should be easy for you to understand as well.

One extremely useful way to use comments for yourself is as an outline for your code. If you're not sure how your program is going to turn out, then you can use comments as a way to keep track of what's left to do, or even as a way of tracking the high-level flow of your program. For instance, use comments to outline a function in pseudo-code:

Python

```
from collections import defaultdict

def get_top_cities(prices):
    top_cities = defaultdict(int)

    # For each price range
    # Get city searches in that price
    # Count num times city was searched
    # Take top 3 cities & add to dict

    return dict(top_cities)
```

These comments plan out `get_top_cities()`. Once you know exactly what you want your function to do, you can work on translating that to code.

Using comments like this can help keep everything straight in your head. As you walk through your program, you'll know what's left to do in order to have a fully functional script. After “translating” the comments to code, remember to remove any comments that have become redundant so that your code stays crisp and clean.

You can also use comments as part of the [debugging](#) process. Comment out the old code and see how that affects your output. If you agree with the change, then don't leave the code commented out in your program, as it decreases readability. Delete it and use version control if you need to bring it back.

Finally, use comments to define tricky parts of your own code. If you put a project down and come back to it months or years later, you'll spend a lot of time trying to get reacquainted with what you wrote. In case you forget what your own code does, do Future You a favor and mark it down so that it will be easier to get back up to speed later on.

## When Writing Code for Others

People like to skim and jump back and forth through text, and reading code is no different. The only time you'll probably read through code line by line is when it isn't working and you have to figure out what's going on.

In most other cases, you'll take a quick glance at variables and function definitions in order to get the gist. Having comments to explain what's happening in plain English can really assist a developer in this position.

Be nice to your fellow devs and use comments to help them skim through your code. Inline comments should be used sparingly to clear up bits of code that aren't obvious on their own. (Of course, your first priority should be to make your code stand on its own, but inline comments can be useful in this regard.)

If you have a complicated method or function whose name isn't easily understandable, you may want to include a short comment after the `def` line to shed some light:

Python

```
def complicated_function(s):
    # This function does something complicated
```

This can help other devs who are skimming your code get a feel for what the function does.

For any public functions, you'll want to include an associated docstring, whether it's complicated or not:

Python

```
def sparsity_ratio(x: np.array) -> float:
    """Return a float

    Percentage of values in array that are zero or NaN
    """
```

This string will become the `.__doc__` attribute of your function and will officially be associated with that specific method. The [PEP 257 docstring guidelines](#) will help you to structure your docstring. These are a set of conventions that developers generally use when structuring docstrings.

The PEP 257 guidelines have [conventions for multiline docstrings](#) as well. These docstrings appear right at the top of a file and include a high-level overview of the entire script and what it’s supposed to do:

Python

```
# -*- coding: utf-8 -*-
"""A module-level docstring

Notice the comment above the docstring specifying the encoding.
Docstrings do appear in the bytecode, so you can access this through
the ``__doc__`` attribute. This is also what you'll see if you call
help() on a module or any other Python object.
"""
```

A module-level docstring like this one will contain any pertinent or need-to-know information for the developer reading it. When writing one, it’s recommended to list out all classes, exceptions, and functions as well as a one-line summary for each.

## Python Commenting Worst Practices

Just as there are standards for writing Python comments, there are a few types of comments that don’t lead to Pythonic code. Here are just a few.

### Avoid: W.E.T. Comments

Your comments should be D.R.Y. The acronym stands for the programming maxim “Don’t Repeat Yourself.” This means that your code should have little to no redundancy. You don’t need to comment a piece of code that sufficiently explains itself, like this one:

Python

```
return a # Returns a
```

We can clearly see that `a` is returned, so there’s no need to explicitly state this in a comment. This makes comments W.E.T., meaning you “wrote everything twice.” (Or, for the more cynical out there, “wasted everyone’s time.”)

W.E.T. comments can be a simple mistake, especially if you used comments to plan out your code before writing it. But once you’ve got the code running well, be sure to go back and remove comments that have become unnecessary.

### Avoid: Smelly Comments

Comments can be a sign of “code smell,” which is anything that indicates there might be a deeper problem with your code. Code smells try to mask the underlying issues of a program, and comments are one way to try and hide those problems. Comments should support your code, not try to explain it away. If your code is poorly written, no amount of commenting is going to fix it.

Let’s take this simple example:

Python

```
# A dictionary of families who live in each city
mydict = {
    "Midtown": ["Powell", "Brantley", "Young"],
    "Norcross": ["Montgomery"],
    "Ackworth": []
}

def a(dict):
    # For each city
    for p in dict:
        # If there are no families in the city
        if not mydict[p]:
            # Say that there are no families
            print("None.")
```



This code is quite unruly. There’s a comment before every line explaining what the code does. This script could have been made simpler by assigning obvious names to variables, functions, and collections, like so:

Python

```
families_by_city = {
    "Midtown": ["Powell", "Brantley", "Young"],
    "Norcross": ["Montgomery"],
    "Ackworth": [],
}

def no_families(cities):
    for city in cities:
        if not families_by_city[city]:
            print(f"No families in {city}.")
```

By using obvious naming conventions, we were able to remove all unnecessary comments and reduce the length of the code as well!

Your comments should rarely be longer than the code they support. If you’re spending too much time explaining what you did, then you need to go back and refactor to make your code more clear and concise.

## Avoid: Rude Comments

This is something that’s likely to come up when working on a development team. When several people are all working on the same code, others are going to be going in and reviewing what you’ve written and making changes. From time to time, you might come across someone who dared to write a comment like this one:

Python

```
# Put this here to fix Ryan's stupid-a** mistake
```

Honestly, it’s just a good idea to not do this. It’s not okay if it’s your friend’s code, and you’re sure they won’t be offended by it. You never know what might get shipped to production, and how is it going to look if you’d accidentally left that comment in there, and a client discovered it down the road? You’re a professional, and including vulgar words in your comments is not the way to show that.

## How to Practice Commenting

The simplest way to start writing more Pythonic comments is just to do it!

Start writing comments for yourself in your own code. Make it a point to include simple comments from now on where necessary. Add some clarity to complex functions, and put a docstring at the top of all your scripts.

Another good way to practice is to go back and review old code that you’ve written. See where anything might not make sense, and clean up the code. If it still needs some extra support, add a quick comment to help clarify the code’s purpose.

This is an especially good idea if your code is up on GitHub and people are forking your repo. Help them get started by guiding them through what you’ve already done.

You can also give back to the community by commenting other people’s code. If you’ve downloaded something from GitHub and had trouble sifting through it, add comments as you come to understand what each piece of code does.

“Sign” your comment with your initials and the date, and then submit your changes as a pull request. If your changes are merged, you could be helping dozens if not hundreds of developers like yourself get a leg up on their next project.

## Conclusion

Learning to comment well is a valuable tool. Not only will you learn how to write more clearly and concisely in general, but you’ll no doubt gain a deeper understanding of Python as well.

Knowing how to write comments in Python can make life easier for all developers, including yourself! They can help other devs get up to speed on what your code does, and help you get re-acquainted with old code of your own.



By noticing when you’re using comments to try and support poorly written code, you’ll be able to go back and modify your code to be more robust. Commenting previously written code, whether your own or another developer’s, is a great way to practice writing clean comments in Python.

As you learn more about documenting your code, you can consider moving on to the next level of documentation. Check out our tutorial on [Documenting Python Code](#) to take the next step.

Mark as Completed

Share

Watch Now

This tutorial has a related video course created by the Real Python team. Watch it together with the written tutorial to deepen your understanding: [Writing Comments in Python](#)


## About **Jaya Zhané**





Jaya is an avid Pythonista and writes for Real Python. She's a Master's student at Georgia Tech and is interested in data science, AI, machine learning and natural language processing.

[» More about Jaya](#)

Each tutorial at Real Python is created by a team of developers so that it meets our high quality standards. The team members who worked on this tutorial are:

[Aldren](#)

[Brad](#)

[Joanna](#)

## What Do You Think?

Rate this article:

LinkedIn

Twitter

Bluesky

Facebook

Email

What’s your #1 takeaway or favorite thing you learned? How are you going to put your newfound skills to use? Leave a comment below and let us know.

**Commenting Tips:** The most useful comments are those written with the goal of learning from or helping out other students. [Get tips for asking good questions](#) and [get answers to common questions in our support portal](#).

Looking for a real-time conversation? Visit the [Real Python Community Chat](#) or join the next [“Office Hours” Live Q&A Session](#). Happy Pythoning!

## Keep Learning

Related Topics: [basics](#) [best-practices](#)

Recommended Video Course: [Writing Comments in Python](#)

Related Tutorials:

- [Documenting Python Code: A Complete Guide](#)
- [Variables in Python: Usage and Best Practices](#)
- [Python Booleans: Use Truth Values in Your Code](#)
- [Basic Data Types in Python: A Quick Exploration](#)
- [11 Beginner Tips for Learning Python Programming](#)

### Learn Python

[Start Here](#)  
[Learning Resources](#)  
[Code Mentor](#)  
[Python Reference](#)  
[Support Center](#)

### Courses & Paths

[Learning Paths](#)  
[Quizzes & Exercises](#)  
[Browse Topics](#)  
[Workshops](#)  
[Books](#)

### Community

[Podcast](#)  
[Newsletter](#)  
[Community Chat](#)  
[Office Hours](#)  
[Learner Stories](#)

### Membership

[Plans & Pricing](#)  
[Team Plans](#)  
[For Business](#)  
[For Schools](#)  
[Reviews](#)

### Company

[About Us](#)  
[Team](#)  
[Sponsorships](#)  
[Careers](#)  
[Press Kit](#)  
[Merch](#)



[Privacy Policy](#) · [Terms of Use](#) · [Security](#) · [Contact](#)

♥ Happy Pythoning!

© 2012–2025 DevCademy Media Inc. DBA Real Python. All rights reserved.  
REALPYTHON™ is a trademark of DevCademy Media Inc.

