

You have 2 free stories left this month. [Sign up and get an extra one for free.](#)

Auto formatters for Python 🧑🏻💻 🤖



Kevin Peters

Jun 3, 2018 · 9 min read ★

Black
autoper8
yapf

This post can be found also on my personal blog here: <https://www.kevinpeters.net/auto-formatters-for-python>

As you can see in the recent [Stack Overflow developer survey](#), Python is one of the most popular programming languages available. With the recent rise of good auto formatters for Code like [gofmt](#) for Golang or [prettier](#) for JavaScript, there is a good question to ask: Which auto formatter to use with Python Code? An auto formatter is a tool which will format your code in a way it complies to the tool or any other standard it set.

First of all, we need to make sure that we know the reason automatic formatting tools exist. Prettier is an auto formatter for JavaScript code. It is used by many big

companies like [Facebook](#), Paypal, Algolia, Yelp, Discord and many others which you can find [here](#). Reasons for this are:

- You do not need a style guide for low-level problems since the auto formatter deals with those problems
- This directly reduces the number of discussions about unnecessary things and let the developers focus on writing actual code
- It will also help with onboarding developers on the code base because the style of the code is consistent
- Less merge conflicts since the style will almost always be the same

Those reasons are really good arguments for using auto formatters. It will save engineers time and company time because developers will be more productive.

For Python there are three solutions out there:

autopep8 — [GitHub](#)

autopep8 is an auto formatter built and open-sourced and made [by several developers](#). It is maybe the most used right now since it is one of the oldest tools out there. It uses [pycodestyle](#) to analyze which parts of your code do not fit to the pep-guidelines and will try to fix them. There is a list of features listed in the README of the repository on which pep guidelines are supported. You can find this list [here](#). The tool will also do small additional checks. As of the time of writing the repository had around 2200 stars (June 2018).

yapf — [GitHub](#)

Yet another Python formatter is another tool which is produced and maintained by none other than [Google](#). It has ~7200 stars (June 2018) on GitHub and follows a different mindset in comparison to autopep8. It will not make code compliant with pep guidelines or try to fix linting issues. It will just format the code. This requires manual work then to make the code look nice which wastes developer time. Other than that yapf is really configurable. It includes defaults for pep8, Google, Facebook and Chromium styling. You can also change a lot of style rules. More information can be found in the [Knobs section](#) of the README. The tool also offers an online version where you can try out the formatting. It can be found [here](#).

black — [GitHub](#)

The last popular auto formatter which is considered in this blog article. It is an initiative of [Łukasz Langa](#) who is a Python Core Developer. The tool is used by different open source libraries like [Fabric 2](#) and [pytest](#). It has around 3800 stars on GitHub (June 2018) and the main incentive is to not have that many options so you do not even have to discuss the options. This mindset is also followed by prettier, a big JavaScript auto formatter.

. . .

All of these tools can be found on [PyPi](#) and be installed via pip or similar Python package managers and can be used on the command line which makes them cross-environment friendly. You should also consider a tool like pre-commit or [husky](#) and [lint-staged](#) for an automatic pre-commit hook which works in every environment and which is easy to install. With this pre-commit hook, unformatted code will never be pushed from the development machine to the remote repository. For the comparison of yapf, I will use the Facebook, Google and pep8 setting since the Chromium setting does not get that much usage (2 spaces indent is a reason for this).

In all examples, I will run the default configuration for each tool. For yapf, this is not possible that easy since there are different base configurations which I wanted to try. Those are pep8, Google and Facebook.

In our first example, we will just show a function call inside a model which is normally too long for a line and see how the auto formatters are restructuring the code here.

```
1 class Basket():
2     reference = models.CharField(
3         _('Project reference'), max_length=100, null=True)
```

```
1 class Basket():
2     reference = models.CharField(
3         _("Project reference"), max_length=100, null=True
4     )
```

```
1 class Basket():
2     reference = models.CharField(
3         _('Project reference'), max_length=100, null=True
4     )
```

Left is the base code, yapf (Google, pep8) did not make any changes, the code in the middle is formatted by black and the right one is formatted by yapf (Facebook)

You can see that yapf (Facebook) and black are trying to convert the code to multi-line brackets while the rest do not do this. A difference between black and yapf (Facebook) is that strings also get formatted in black which you will see later too. Black uses double quotes for every string except for strings where double quotes are included.

. . .

The next example is again a real-world example. Here we have a tuple which includes a nested tuple. It works more or less like an enumeration. Also, the `_` is used to translate the strings in the application.

```
1 class Basket():
2     PRICING_STATUS_CHOICES = (
3         (NO_REQUEST,
4          _("No Request - there was no manual pricing requested yet")),
5         (WAITING_FOR_MANUAL_PRICING,
6          _(("Waiting For Pricing - the basket needs someone to "
7            "set a manual price for one or multiple lines"))),
8         (MANUALLY_PRICED,
9          _("Manually Priced - the basket has been priced manually")),
10    )
```

Base code, autopep8 and yapf (Google, pep8)

You can see that this was the input for the auto formatters but autopep8, yapf (Google) and yapf (pep8) did not change anything in the code.

```
1 class Basket():
2     PRICING_STATUS_CHOICES = (
3         (
4             NO_REQUEST,
5             _("No Request - there was no manual pricing requested yet")
6         ),
7         (
8             WAITING_FOR_MANUAL_PRICING,
9             _((
10                 "Waiting For Pricing - the basket needs someone to "
11                 "set a manual price for one or multiple lines"
12             ))
13         )
14     ),
15     (
16         MANUALLY_PRICED,
17         _("Manually Priced - the basket has been priced manually")
18     ),
19 )
20 )
```

black and yapf (Facebook)

Black and yapf (Facebook) gave the same result in the end since the input was formatted with double quotes already. Also, both of the formatters split up the lines a bit.

. . .

The next example is testing a function with a lot of parameters. In reality, you should never do this, and instead pass an object as a parameter if you have more than three parameters. However, this is a great example for testing the auto formatters.

This time we get mostly different results. If you are trying to decide on an auto formatter, look at the next two examples. They will show you the real differences between the tools.


```
1 def function_with_really_long_name(normal_variable, another_normal_variable, configuration=None, test_number=1, test_text='dwadawa', test_tuple=('LoremIpsum', 21)):  
2     return None
```

Base code

```
1 def function_with_really_long_name(  
2     normal_variable,  
3     another_normal_variable,  
4     configuration=None,  
5     test_number=1,  
6     test_text='dwadawa',  
7     test_tuple=(  
8         'LoremIpsum',  
9         21)):  
10    return None
```

autopep8


Autopep8 is formatting the parameters below each other and also starting a new line since it analyzed that there are too many parameters to fit. With the default value for `test_tuple` it got some problems. The code is completely pep8-compliant but the tuple definition looks quite odd. We can also see again that closing brackets are not moved to a new line but instead it will do the minimum work to make the code pep8-compliant.



```
1 def function_with_really_long_name(  
2     normal_variable,  
3     another_normal_variable,  
4     configuration=None,  
5     test_number=1,  
6     test_text="dwadawa",  
7     test_tuple=( "LoremIpsum", 21 ),  
8 ):  
9     return None
```

black

Black is formatting the code similar to autopep8 but is moving the closing brackets to a new line and also does not format the tuple in an unnecessary way.



```
1 def function_with_really_long_name(  
2     normal_variable,  
3     another_normal_variable,  
4     configuration=None,  
5     test_number=1,  
6     test_text='dwadawa',  
7 ):
```

```

7 test_tuple=( 'LoremIpsum', 21)
8 ):
9 return None

```

yapf (Facebook)

Yapf with the Facebook setting is formatting very similar to black here but is not changing the quotes here again. The only difference which happened.

```

1 def function_with_really_long_name(normal_variable,
2                                     another_normal_variable,
3                                     configuration=None,
4                                     test_number=1,
5                                     test_text='dwadawa',
6                                     test_tuple=( 'LoremIpsum', 21)):
7     return None

```

yapf (Google), yapf (pep8)

Yapf with the Google and pep8 setting will put the parameters below each other but will try to put the first parameter on the same line where the function begins.

. . .

The last example which I will list in this article is quite an edge-case. First of all, we will create a namedtuple and then try to put tuples of this type into a list. This list will then work as parameters for a generator expression with specific filtering.

```

1 def generator_expression():
2     fruit = collections.namedtuple('Fruit', ('name', 'size', 'price', 'super_long_property_in_tuple'))
3     fruits = [
4         Fruit(name='apple', size=5, price=10.50, super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'),
5         Fruit(name='banana', size=7, price=10.50, super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'),
6         Fruit(name='orange', size=6, price=10.50, super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'),
7         Fruit(name='kiwi', size=1, price=10.50, super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.')
8     ]
9     complicated_fruits_filtered = [fruit for fruit in fruits if fruit.price >= 10 and size <= 5]

```

Base code

```

1 def generator_expression():
2     Fruit = collections.namedtuple(
3         'Fruit', ('name', 'size'
4                 'price super_long_property_in_tuple'))
5     fruits = [
6         Fruit(
7             name='apple',
8             size=5,
9             price=10.50,
10            super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'),
11        Fruit(
12            name='banana',
13            size=7,
14            price=10.50,
15            super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'),
16        Fruit(
17            name='orange',
18            size=6,
19            price=10.50,
20            super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'),
21        Fruit(
22            name='kiwi',
23            size=1,
24            price=10.50,
25            super_long_property_in_tuple='super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.')]
26     complicated_fruits_filtered = [
27         fruit for fruit in fruits if fruit.price >= 10 and size <= 5]

```

autopep8

Autopep8 also tries here to do the minimal work. The namedtuple will be split into multiple lines. The arguments for the names are also split into two lines. Other than that it will also try to leave the text on the same line in the `super_long_property_in_tuple` definition. The generator expression just gets put onto the next line.

```

1 def generator_expression():
2     Fruit = collections.namedtuple(
3         "Fruit", ("name", "size" "price super_long_property_in_tuple")
4     )
5     fruits = [
6         Fruit(
7             name="apple",
8             size=5,
9             price=10.50,
10            super_long_property_in_tuple="super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.",
11        ),
12        Fruit(
13            name="banana",
14            size=7,
15            price=10.50,
16            super_long_property_in_tuple="super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.",
17        ),
18        Fruit(
19            name="orange",
20            size=6,
21            price=10.50,
22            super_long_property_in_tuple="super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.",
23        ),
24        Fruit(
25            name="kiwi",
26            size=1,
27            price=10.50,
28            super_long_property_in_tuple="super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.",
29        ),
30    ]
31     complicated_fruits_filtered = [
32         fruit for fruit in fruits if fruit.price >= 10 and size <= 5
33     ]

```

black

Black is formatting strings to double quotes. The namedtuple is split up into three lines where the second line is just the important data. The array splitting happens as usual and brackets are moved a lot and not put behind the last elements. What is really interesting is that the string on line 10, for example, is not moved to the next line.

The generator expression is also split up similar to autopep8 but the closing bracket was also moved to the next line.

```

1 def generator_expression():
2     Fruit = collections.namedtuple(
3         'Fruit', ('name', 'size'
4                 'price super_long_property_in_tuple')
5     )
6     fruits = [
7         Fruit(
8             name='apple',
9             size=5,
10            price=10.50,
11            super_long_property_in_tuple=
12                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
13        ),
14        Fruit(
15            name='banana',
16            size=7,
17            price=10.50,
18            super_long_property_in_tuple=
19                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
20        ),
21        Fruit(
22            name='orange',
23            size=6,
24            price=10.50,
25            super_long_property_in_tuple=
26                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
27        ),
28        Fruit(
29            name='kiwi',
30            size=1,
31            price=10.50,
32            super_long_property_in_tuple=
33                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
34        )
35    ]
36    complicated_fruits_filtered = [
37        fruit for fruit in fruits if fruit.price >= 10 and size <= 5
38    ]

```

yapf (Facebook)

Yapf with the Facebook configuration also formats the tuples similarly to black but places the long parameter onto a new line. The `super_long_property_in_tuple` is also broken up into a new line. The generator expression looks exactly like the one which was generated by black.

```

1 def generator_expression():
2     Fruit = collections.namedtuple('Fruit',
3                                   ('name', 'size'
4                                   'price super_long_property_in_tuple'))
5     fruits = [
6         Fruit(
7             name='apple',
8             size=5,
9             price=10.50,
10            super_long_property_in_tuple=
11                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
12        ),
13        Fruit(
14            name='banana',
15            size=7,
16            price=10.50,
17            super_long_property_in_tuple=
18                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
19        ),
20        Fruit(
21            name='orange',
22            size=6,
23            price=10.50,
24            super_long_property_in_tuple=
25                'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'
26        )
27    ]
28    complicated_fruits_filtered = [
29        fruit for fruit in fruits if fruit.price >= 10 and size <= 5
30    ]

```

```
24     super_long_property_in_tuple=  
25     'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'  
26 ),  
27     Fruit(  
28         name='kiwi',  
29         size=1,  
30         price=10.50,  
31         super_long_property_in_tuple=  
32         'super long string here also, lorem ipsum, maybe longer than 80 characters to look for pep8 violations here. lorem ipsum.'  
33     )  
34 ]  
35 complicated_fruits_filtered = [  
36     fruit for fruit in fruits if fruit.price >= 10 and size <= 5  
37 ]  
38
```

yapf (Google), yapf (pep8)

Yapf with the Google and pep8 setting will format the code the same in this case. The main difference to the Facebook setting is that the namedtuple is formatted differently, as we saw in previous examples like the parameter splitting.

. . .

Conclusion

All of the formatters are doing a good job at formatting the code. But in my opinion, autopep8 is not really formatting but more or less just trying make your code compliant to pep8. Still, the code might look bad and does not fulfill the requirement of being an auto formatter.

Black and yapf both have their own advantages and disadvantages. Yapf is highly configurable but I would recommend keeping the settings as little as possible. For yapf, the Facebook setting is quite similar to black's but I think it is really weird that, for example, with the default configuration quotes are not formatted by default.

Otherwise, it is just personal preference on which configuration to take or which formatter to choose.

For me, personally, I will choose black for future projects since it is close to prettier in its approach. Yapf might have the bigger backing but I think this will change over time.

Also, another note: Use one of the auto formatters presented in this article. Your development team will save so much time. It will be incredible. Also, your personal code will look the same throughout different projects and repositories and everyone will be much happier. And remember to not nitpick so much about code style. People are opinionated, but here efficiency and less communication are far more important than opinions.

• • •

This blog post was written in cooperation with the company 3YOURMIND. We are looking for developers in Berlin. You can find openings here. We are a 3D Printing startup with a lot of cool developers using *Vue.js*, *Django REST*, *Java*, *Docker* and many more cutting-edge frameworks and libraries to change the enterprise 3D printing world.

Thanks for reading this. You rock 🙌

If you have any feedback or want to add something to this article just comment here. You can also follow me on twitter or visit my personal site to stay up-to-date with my blog articles and many more things.

Thanks to James O'Shea.

Programming

Python

Clean Code

Software Development

Software Engineering

Medium

About Help Legal

Get the Medium app

