Writing (good) code in Python

Why should I read this document?

To code, you need to not only know the language and implementation but also how to write clean, readable code and the proper tools to do so. There are softwares such as specialized text editors for programming and IDEs provide important tools such as auto-indentation, syntax highlighting, auto-complete, linter, debugger, and of course, dark themes. The Python community has mutually decided upon some rules to be considered as good practice while writing code. Nearly every organization that uses python mandates that the code should follow these proposals.

Python Enhancement Proposal (PEP)

PEPs are suggestions for improvements to Python by the Python community. Its main objective is proposing new features, taking community feedback on issues and most importantly, documenting Python design decisions.

The Zen of Python (PEP 20)

Writing programs that do what they are supposed to do is just one component of being a good Python programmer. It is also important to write clean code that is easily understood, even weeks after you've written it.

One way of doing this is to follow the Zen of Python, a somewhat tongue-in-cheek set of principles that serves as a guide to programming the Pythonic way.

Running "`import this`" in Python will print the Zen of Python, given below. Don't worry if you don't understand any of this. As you continue exploring the language, you will automatically be able to relate to this more and more.

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
*Although never is often better than *right* now.*
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Some lines in the Zen of Python may need more explanation.

- Explicit is better than implicit: It is best to spell out exactly what your code is doing. This is why adding a numeric string to an integer requires explicit conversion, rather than having it happen behind the scenes, as it does in other languages.
- Flat is better than nested: Heavily nested structures (lists of lists, of lists, and on and on…) should be avoided.
- Errors should never pass silently: In general, when an error occurs, you should output some sort of error message, rather than ignoring it.
- There are 20 principles in the Zen of Python, but only 19 lines of text. The 20th principle is a matter of opinion, but one interpretation is that the blank line means "use whitespace".

Style Guide for Python Code (PEP 8)

PEP 8 is a style guide on the subject of writing readable code. It contains a number of guidelines, few of which are summarized here:

- modules should have short, all-lowercase names;
- class names should be in the CapWords style (called PascalCase);
- most variables and function names should be lowercase_with_underscores (called snake_case);
- constants (variables that never change value) should be CAPS_WITH_UNDERSCORES;
- names that would clash with Python keywords (such as 'class' or 'if') should have a trailing underscore
- lines shouldn't be longer than 80 characters;
- 'from module import *' should be avoided;
- there should only be one statement per line.

The most important advice in the PEP is to ignore it when it makes sense to do so. Don't bother with following PEP suggestions when it would cause your code to be less readable; inconsistent with the surrounding code; or not backwards compatible. However, by and large, following PEP 8 will greatly enhance the quality of your code.

> *"Programs must be written for people to read, and only incidentally for machines to execute."*
>
> *— Harold Abelson, Structure and Interpretation of Computer Programs*

> *"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."*
>
> *— John Woods*

To get a better understanding of PEP-8, and see examples for each rule, read through this.

Learning and writing with PEP 8

Obviously, we can't be expected to keep the website open while we are coding and keep referring to it. That's where the IDE/Text Editor comes into the picture. They have various features to help you in your development regime. You can (and should) enable a "linter", which is responsible for highlighting the part of your code that deviates from specific rules, like PEP8.

What is an IDE, and how is it different from a text editor?

IDE stands for Integrated Development Environment. It is a software which typically consists of a code editor, build automation tools and a debugger. However, text editors usually lack debuggers and build automation. They are lighter (use less memory) when compared to IDEs in exchange for the features. Some modern IDEs also have a class browser, object browser and a class hierarchy diagram for Object Oriented Programming (OOP). For example, Code::Blocks is an IDE while gedit is a text editor.

Editors and IDE for Python

Python comes with an inbuilt Integrated Development and Learning Environment(IDLE). You must NOT use it, as it lacks basic text editor features such as auto-complete, line numbering and customizability.

Sublime Text (text editor) - link

Sublime Text is a cross-platform and lightweight text editor and is the most recommended text editor owing to its many features like plugin support, goto definition, multiple selections, command palette etc. You can even turn it into a powerful IDE for python by installing plugins like Anaconda, which provides a linter, and SublimeGit, to manage your version control. Also, don't worry about the free "trial", it will never expire.

PyCharm (IDE) - link

PyCharm is a cross-platform IDE which provides code analysis, a graphical debugger, integration with version control systems (like Git), and supports web development with Django.

Atom (text editor) - link

Atom is an open source and cross-platform code editor with support for plug-ins and embedded version control (Git).

Visual Studio Code (text editor) - link

Visual Studio Code is a code editor developed by Microsoft for Windows, Linux and MacOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring.

There's also various command line editors, like Vim and Emacs, which are supposed to be run on a terminal, and interacted to via a keyboard.

Can't decide?

It is better to start with an editor (any of the above will do) for now, and after you have had some experience in programming, you can begin testing the others to see what suits your needs the best.