# How to Write Code Like a Pro

## Approaches and techniques that help you to write code to professional standards

### Background

I have been writing code for 20 years and over that time I have established a set of 10 principles that I believe programmers, developers and data scientists can adopt to help them to write their programme code to professional standards.

These approaches are generally applicable to any software development environment but as all of my coding these days is in the Python language in the VS Code development environment I have focused on those tools for specific examples.

## Principles, Tools and Techniques

### Linting

The first easy win to help in writing professional code is to use linting.

“Linting highlights syntactical and stylistic problems in your Python source code, which often helps you identify and correct subtle programming errors or unconventional coding practices that can lead to errors.” ([https://code.visualstudio.com/docs/python/linting#](https://code.visualstudio.com/docs/python/linting))

To start linting in VS Code go to the command palette with Ctrl+Shift+P and type “Select Linter”, then chose the linter you want to use.

There are a range of linters available and the choice will depend on individual and team preference but my personal favourite is “pylint” because I like way in which it presents issues in the code and because it is easy to configure.

Once linting is on VS Code will display its suggestions in the problems window which will update dynamically as you write new code and improve existing code –

A screenshot of a computer

Description automatically generated with medium confidence

Resolving all of the pylint suggested errors and warnings will quickly enable your code to reflect professional, consistent standards and to follow best practice.

### Comments, Type Hints and Documentation

Let’s start with type hints. Python is a “dynamically typed” language. The type of a variable is not required and most online examples omit the type of variables, parameters and return types.

The drawback is that the client or caller cannot easily know what type the function is expecting.

However, If the variable types are optionally added to the code the readability and understandability increase dramatically because the client or caller will instantly know what type the function is expecting …

A screenshot of a computer

Description automatically generated with medium confidence

Once type hints are included the next stage is well commented and documented code which is one of the things that sets professional code apart.

In addition to improving readability and maintainability as you add comments, type hints and documentation it makes you think about your code from a different perspective leading to reflective self-feedback and improvement.

One the docString extension is installed in VS Code just press return underneath a function declaration, enter three double quotes (“””) and then fill out the stub that is created for you …

Text

Description automatically generated

It is also important to leave general comments in the code to increase understandability and maintainability and wrapping comment blocks in a region enables them to be neatly folded in the VS Code editor …

Text

Description automatically generated

When type hints and docStrings have been completed throughout a module it is then very easy to create a full set of professional looking documentation. Simply invoke pydoc as follows …

python -m pydoc -w "..\lib\crypto\_tools.py"

… and a documentation web page will be automatically created …

Graphical user interface, text, application

Description automatically generated

### Project Structure

Many smaller projects can get away with being created in a single folder that contains all of the code and configuration files necessary to run the project.

However, it does not take long before a single folder can become an unstructured dumping ground leading to messy and unprofessional projects.

I have seen several recommendations for a standard folder layout for Python projects and concluded that it does not matter which one is chosen so long as it provides a sensible, logical, intuitive, discrete and consistent organisation of project resources.

This is the standard I have adopted for my projects …

Text

Description automatically generated

The “data” sub-folder contains any data files relating to the project and I commonly add “in” and “out” folders if my project cleans or transforms data.

“docs” is where I store the documentation created by pydoc from the docStrings and a batch to invoke pydoc to enable one-click documentation production.

“keys” was a special project for this project which created security keys demonstrating that I am not averse to extending my standard approach based on the needs of the project.

“lib” is where I store any re-usable code libraries. In this case I moved all the code that had potential future re-use value into crypto\_tools.py and refactored it to maximise usability and maintainability.

“notebooks” is where all Jupyter Notebooks are separated and stored. I usually use Notebooks to create a sample user interface and to demonstrated and show off how a project works and can be used.

“src” is where I store any other Python code that is not a reusable library.

“unittests” is where all of the pytest unit tests are stored. In a medium-large project there could be a lot of unit test files and these really need moving to a discrete location to maintain tidiness. This requires a bit of additional configuration that I will document in a future article.

A well organised and structured project instantly adds professional kudos to programme code.

### Unit Testing

Spending time developing unit tests is critical to coding like a professional and my personal preference for a unit testing framework is pytest.

Once the unit tests have been created (see <https://code.visualstudio.com/docs/python/testing> for more details) simply use the flask icon inside VS Code to discover all the unit tests and then click play to execute them all –

Text

Description automatically generated

They will light up green if everything works OK or red if any unit tests fail.

The power of comprehensive unit testing is that future changes and updates can be made in the confidence that if anything is inadvertently broken a single-click test run will instantly highlight the problems.

In this way code can be developed and maintained professionally and current and future code quality will be high.

### Object Oriented Programming

The 4 main concepts in Object Oriented Programming (OOP) are

* Inheritance - a class inherits the properties and methods from another class
* Encapsulation – data is hidden and secured inside its class through private properties
* Polymorphism - methods can have the same name with a different implementation - think len(int) vs. len(list)
* Abstraction – The automatic enforcement of standard interfaces – think .fit() and .fit\_transform() in scikit-learn

A detailed explanation with some Python examples can be found here - <https://www.analyticsvidhya.com/blog/2020/09/object-oriented-programming/>.

There are many benefits to OOP vs. traditional procedural programming. including code re-use, maintainability, security, productivity, easier trouble shooting etc.

The key advantage of OOP for me however is that the consumers and clients of classes and objects have a much more intuitive and usable interface leading to front-end code that is more compact, maintainable and readable.

For example, the following code snippet shows an intuitive and simple interface for one way hashing which may otherwise have been messy and complex -

one\_way\_hash = OneWayHash(hashing\_algorithm="SHA256")

hash\_value\_1 = one\_way\_hash.hash\_data("Hello World")

hash\_value\_2 = one\_way\_hash.hash\_data("Hello Universe")

That is the power of OOP and why I always write my code as classes and objects and this is one of the key approaches in increasing the professionalism of programme code.

### Code Duplication and Refactoring

The duplication of the same or very similar code leads to projects that are prone to error. If a similar piece of code has been repeated 10 times in a project and then a bug needs fixing or an enhancement adding it needs doing 10 times which is inefficient and laborious and provides 10 opportunities for mistakes.

Consider these simple functions –

STRING\_ENCODING : str = "ascii"

def bytes\_to\_str(data : bytes) -> str:

    return data.decode(STRING\_ENCODING)

def str\_to\_bytes(data : str) -> bytes:

    return bytes(data, STRING\_ENCODING)

Each one is only replacing a single line of code so why bother? Well, this code snippet is taken from a project that contained 100s of instances of the decode and encode and at one point in the project the encoding had to be changed from “utf-8” to “ascii”.

This required many changes and somehow a couple of them were missed leading to bugs and errors that were not spotted until after the code went into production.

By making the change to pulling the instances out into two simple functions the code looked cleaner, the duplication was eradicated and any future changes to the encoding can be made quickly and confidently.

In programming parlance this is known as the “DRY” method – “Don’t Repeat Yourself” (see <https://en.wikipedia.org/wiki/Don%27t_repeat_yourself> for more details).

### Refactoring

Refactoring is an iterative approach to reviewing and improving existing code until it is as clean, neat and professional as you can make it.

Here are some questions and consideration to help with the review process …

1. Can several lines of code be replaced with fewer lines?
2. Conversely does the code need to be a little bit more verbose to improve readability.
3. Have bespoke code been written where an existing library already can do the same job?
4. Can repetitive code be eliminated?
5. Could several associated functions and data be re-written as a class (OOP)?
6. Could the code be extended to enhance the future proofing and re-usability?
7. Have exceptions and error handling been considered and included?
8. Have “pythonic” approaches like using lambda functions, list comprehension etc. been fully utilised?
9. Is the code efficient and fast to execute?
10. Is the code re-runnable and re-usable?

Considering these questions and becoming obsessive about iteratively refactoring your code until it is close to being perfect is one of the key techniques that will help you to code like a pro.

### Building Libraries

There is always pressure from employers and customers to work quickly which can sometimes lead to sloppy coding, but you can work quickly without sacrificing quality by building reusable code libraries.

My personal approach is to maintain two libraries with slightly different purposes.

The first is called “Sample Code”. It is a dumping ground for all the useful code snippets I come across online and in books that I know I will want to refer back to in future and then not be able to find later on!

My second library is my “Utilities” library. To qualify for the library code has to be re-factored, tidied, tested, documented and structured in such a way that it will be generically useful and reusable in future projects.

Here is an example. I needed some synthetic data to test a classification algorithm. Google soon helped me track the code down but it was a bit messy and undocumented. After a bit of extra work my utilities library gained a useful new method as follows –

def make\_classification\_dataframe(n\_samples : int = 10000, n\_features : int = 25, n\_classes : int = 2, n\_clusters\_per\_class : int = 2, feature\_name\_prefix : str = "feature\_", target\_name : str = "target", random\_state : int = 42) -> pd.DataFrame:

    """ Creates a data frame of sample data suitable as input to a classification machine learning algorithm consisting of the specified number of features and a target with the specified number of classes.

        Args:

            n\_samples (int, optional): The number of data points to generate. Defaults to 10000.

            n\_features (int, optional): The number of features in the data. Defaults to 25.

            n\_classes (int, optional): The number of classes in the target. Defaults to 2.

            n\_clusters\_per\_class (int, optional): The number of clusters. Defaults to 2.

            feature\_name\_prefix (str, optional): The prefix for the feature names in the data. Defaults to "Feature" which generates "Feature 1", "Feature 2", ... , "Feature n".

            target\_name (str, optional): The prefix for the name of the target. Defaults to "Target".

            random\_state (int, optional): The seed for the random state. Defaults to 42.

        Returns:

            pd.DataFrame: The generated DataFrame.

        Example:

            >>> df\_data = make\_classification\_dataframe(n\_samples=N\_SAMPLES, n\_features=N\_FEATURES, n\_classes=N\_CLASSES, n\_clusters\_per\_class=N\_CLUSTERS\_PER\_CLASS, feature\_name\_prefix=FEATURE\_NAME\_PREFIX, target\_name=TARGET\_NAME, random\_state=RANDOM\_STATE)

            >>> X = df\_data.drop([TARGET\_NAME], axis=1).to\_numpy()

            >>> y = df\_data[TARGET\_NAME].to\_numpy()

            >>> df\_data.head()

        """

    X, y = make\_classification\_data(n\_samples=n\_samples, n\_features=n\_features, n\_classes=n\_classes, n\_informative = n\_classes \* n\_clusters\_per\_class, random\_state=random\_state)

    feature\_names = [feature\_name\_prefix + str(v) for v in np.arange(1, n\_features+1)]

    return pd.concat([pd.DataFrame(X, columns=feature\_names), pd.DataFrame(y, columns=[target\_name])], axis=1)

The only other thing you need to do is import your library into future projects as follows -

import sys

sys.path.insert(1, r'C:\Users\GHarrison\Python Projects\ Utilities')

from synth\_data\_tools import make\_classification\_data

sys.path.insert adds the utilities folder to the Python path for the project and then the make\_classification\_data is imported and can be used.

Building libraries will help you to code like a pro. The act of refactoring code will improve your knowledge and understanding and enable you to code with both speed and high quality.

### Write Blogs

The protégé effect is a psychological phenomenon where teaching … or preparing to teach information to others helps a person learn that information (<https://effectiviology.com/protege-effect-learn-by-teaching/>).

That is just one of the many benefits of regularly blogging about coding.

<https://grahamharrison-86487.medium.com/>.

Preparing a blog involves reviewing code with a renewed critical eye; after all you do not want any mistakes to make it onto a public article!

Also, the act of explaining your code to others will helps to understand it completely and to improve knowledge and expertise in the process.

Trust me on this one – one of your future readers and students will be yourself! In 6 months or a year’s time you will have forgotten all the details of that really useful coding technique you discovered and you will go back to your own articles to help you remember.

Lastly, blogging on a reputable platform like medium and towards data science will help build your professional persona online which help your peers, the programming community and potential future employers to understand your professional standards and capabilities.

### Reading and Challenges

Read as much relevant material as you can get your hands to help you improve your professional coding skills and subject knowledge

Sign up for mailing lists like Real Python (<https://realpython.com/>) and make sure you join medium.com …

<https://grahamharrison-86487.medium.com/membership>.

Note: the author will receive a proportion of the fees if you sign up using this link.

… then put the app front-and-centre on your smart phone so you can be reading about your subject every time you are stood in a shopping queue or are waiting for the kettle to boil …

Graphical user interface

Description automatically generated

Also sign up to the “Towards Data Science” podcast so you can be improving your professional skills whilst you are driving or sat on the sofa - <https://open.spotify.com/show/63diy2DtpHzQfeNVxAPZgU?si=224fbdb47b6f4c1f>.

In addition to these free or low cost resources it is also worth paying real money for a good, old fashioned book. Here is a selection of the books that I have used recently to improve my skills –

<https://www.amazon.co.uk/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1098125975>

<https://www.amazon.co.uk/Python-Machine-Learning-Example-scikit-learn/dp/1800209711>

<https://www.amazon.co.uk/Deep-Learning-Coders-fastai-PyTorch/dp/1492045527>

<https://www.amazon.co.uk/Practical-Blockchains-Cryptocurrencies-Application-Applications/dp/1484258924>

The last recommendation of this section is about challenging yourself using the wealth of online tools and resources.

If you want general Python coding challenges you can use a resource like “Python Principles” - <https://pythonprinciples.com/challenges/>.

If you want a bigger data science-type challenge and to measure your performance against your peers, why not pick a dataset you like the look of on Kaggle and see how far up the leader board you can get –

<https://www.kaggle.com/competitions/credit-default-prediction-ai-big-data/leaderboard>

## Conclusion

Becoming a professional coder does involve hard work and dedication but if, like me, coding excites and enthuses you and if you love your subject you are going to get there and in this article I have explored 10 tools and techniques that can accelerate that journey.

As well as writing I have coached and mentored many programmers and if you are considering mentoring why not get in touch with me at [GHarrison@lincolncollege.ac.uk](mailto:GHarrison@lincolncollege.ac.uk).

## Thank you for reading!

If you would like to support the author and 1000's of others who contribute to article writing world-wide by subscribing, please use this link –

<https://grahamharrison-86487.medium.com/membership>. Note: the author will receive a proportion of the fees if you sign up using this link.

And if you enjoyed reading this article, why not check out my other articles at <https://grahamharrison-86487.medium.com/>?

Also, I would love to hear from you to get your thoughts on my other articles or anything else related to data science and data analytics. Please feel free to get in touch on LinkedIn at <https://www.linkedin.com/in/grahamharrison1> or e-mail via [GHarrison@lincolncollege.ac.uk](mailto:GHarrison@lincolncollege.ac.uk).