Coding Standard

CONTENTS

1. Introduction 1

2. Indentation 1

3. Maximum Line Length 2

4. Line Breaks 2

5. Importing Libraries 3

6. Commenting the Code 4

    6.1 Inline Comments 4
    6.2 Block Comments 4
    6.3 Docstrings or Document Strings 5

7. Naming Conventions 5

    7.1 Class Names 5
    7.2 Function and Variables Names 6
    7.3 Function and Method Names 6
    7.4 Constants 6
    7.5 Public and Internal Interfaces 6

8. Conclusion 7

1. Introduction:

Python, being one of the most popular languages today, is known for its simplicity, readability and collaborative use. If you are someone who is very new to the Data Science field or has been around in this field for a while, these Python best practices would help you boost in your career and also make your teammate’s life easier for readability. Coding is an art, so you are the artist who will make wonders if you follow best practices in Python.

2. Indentation:

Python needs to have indentation of the code in order to run successfully. Indentation can be done by using some spaces at the start of the code line. It is a requirement in this language and thus, it’s mandatory to follow the rules of indentation. Below is the example of indentation. In the “for” loop, the condition “if x%2 == 0” is indented by giving 4 spaces after the for loop. If we don’t give these spaces, the code will throw an error. Due to this indentation, it becomes easy to read Python code.

Python Code:

```
for x in range(10):
    if x%2 == 0:
        print("{} is an even number".format(x))
    else:
        print("{} is an odd number".format(x))
```

In Python, indentation is mandatory. For example, in the above code if you write the if statement exactly below the for loop, then the Python code will throw an indentation error. Due to indentation, it’s easier to understand and read the code.

3. Maximum Line Length:

Python was developed in such a way that readability and cleanliness would be at the center of every code. Thus, in Python, the maximum number of characters in one line cannot exceed 79. These are requirements of a standard Python library. Due to this limit, Python codes are often easier to read and you can work on multiple Python files side by side due to this limit.

4. Line Breaks:

Instead of writing a long line or a lengthy line of code, you can split that line across multiple lines to enable better readability. All the expressions can be placed in the parentheses and each expression can be written onto a different line. There can be different types of line breaks.

For example, if you are writing a text and assigning it to a variable, you can use backslashes for breaking up the lines. In the case of formulas, the best method is to write each expression in a separate line. Below is the screenshot of bad practice of writing long lines of code versus the best practice is to do it.

Bad Practice:

```
# Without using line breaks
Total_Cost = Rental_Cost + Furniture_Cost + Electricity_Cost + Resourcing_Cost + Gas_Cost
```

Good Practice:

```
# Use Line breaks
Total_Cost = (Rental_Cost
          + Furniture_Cost
          + Electricity_Cost
          + Resourcing_Cost
          + Gas_Cost)
```

From the above screenshots, it is evident that using line breaks makes it easier to read than writing all the expressions in a single line.

5. Importing Libraries:

When you are working with Python, it's recommended that you import all the Python libraries, classes at the start of the program. This allows the user to check what libraries are being used in the program and see if any dependencies are needed or not. Also, it's not ideal to import a library just above the function call from that library. Ideally, you should import all the libraries in separate lines but if you need to import multiple modules from the same library, it's okay to mention those in the same line.

Bad Practice:

```
# Import your Libraries in same line
import pandas as pd, numpy as np
```

Good Practice:

```
# Import your Libraries in separate lines
import pandas as pd
import numpy as np
# Import multiple modules from the same library in one line
from sklearn.metrics import confusion_matrix, classification_report
```

6. Commenting the Code:

The cornerstone of sound coding procedure is the comment. It's crucial to thoroughly document your Python code and to update all of the comments whenever the code is modified. Complete sentences, preferably in English, should be used while leaving comments. There are three different types of comments used in any programming language:

6.1 Inline Comments

These comments are written on the same line as that of the Python code. These comments are useful when you are performing a certain operation and need to make sure that the code reviewer understands what that specific operation is doing. For example, if you are replacing the missing values from a certain column with 95th percentile value, it's easier to write the comment inline with the function. The comments are started by #. Inline comments are shown in the screenshot below.

Example:

```
# Import your Libraries in same line
import pandas as pd, numpy as np


# Calculate addition of x and y and store in z
z = x + y
```

6.2 Block Comments

These comments typically have one or more paragraphs. These are also started with # but if you need to have comments in multiple lines, then use # in different lines. It's a good idea to give a space after # before you start writing the actual comment.

Example:

```
# This is a comment
# written in
# more than just one line
print("Hello, World!")
```

6.3 Docstrings or Document Strings

These types of comments are typically used when there is an explanation needed. It’s used before a class, module or a function in Python. These comments are surrounded by “””Three Double Quotes”””.

Example:

```
"""
This is the docstring comment usually used to describe a function, class or a module. We can
write the comments in multiple lines by using three double quotes.
"""
print("Hello, World!")
```

7. Naming Conventions:

7.1 Class Names

Class names should normally use the CapWords convention. The naming convention for functions may be used instead in cases where the interface is documented and used primarily as a callable. Note that there is a separate convention for builtin names: most builtin names are single words (or two words run together), with the CapWords convention used only for exception names and builtin constants.

7.2 Function and Variables Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability. Variable names follow the same convention as function names. mixedCase is allowed only in contexts where that's already the prevailing style (e.g. threading.py), to retain backwards compatibility.

7.3 Function and Method Names

Always use **self** for the first argument to instance methods.

Always use **cls** for the first argument to class methods. If a function argument's name clashes with a reserved keyword, it is generally better to append a single trailing underscore rather than use an abbreviation or spelling corruption. Thus **class_** is better than **clss**. (Perhaps better is to avoid such clashes by using a synonym.)

7.4 Constants

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX_OVERFLOW and TOTAL.

7.5 Public and Internal Interfaces

Any backwards compatibility guarantees apply only to public interfaces. Accordingly, it is important that users be able to clearly distinguish between public and internal interfaces. Documented interfaces are considered public, unless the documentation explicitly declares them to be provisional or internal interfaces exempt from the usual backwards compatibility guarantees. All undocumented interfaces should be assumed to be internal.

To better support introspection, modules should explicitly declare the names in their public API using the **__all__** attribute. Setting **__all__** to an empty list indicates that the module has no public API.

Even with **__all__** set appropriately, internal interfaces (packages, modules, classes, functions, attributes or other names) should still be prefixed with a single leading underscore. An interface is also considered internal if any containing namespace (package, module or class) is considered internal. Imported names should always be considered an implementation detail. Other modules must not rely on indirect access to such imported names unless they are an explicitly documented part of the containing module's API, such as **os.path** or a package's **__init__** module that exposes functionality from submodules.

8. Conclusion:

We hope to follow this document as much as we can throughout the entirety of the term project.