Python Coding Conventions.

> "Your code is read much more often than it is written."

This guide follows the conventions of PEP 8, a commonly accepted code style guide for Python. We highly recommend you
to follow this guide since the entire Python community does their best to follow these guidelines.
It really helps to make your code more consistent when working on projects with others and also supports its intelligibility.
This document is meant as a short cheatsheet, summarizing the most important aspects. But you can find a more detailed description here.

1. Indentation

Indentation can lead to syntax errors. The recommendation is therefore to use 4 spaces for indentation:

```
for element in range(0, 5):
    print(element)
```

When you write a big expression, it is best to keep the expression vertically aligned

```
dict_of_people_ages = {
 "peter": 20,
 "paul": 30,
 "marie": 40,
 "max": 18
}
```

Whether to use tabs or spaces for indentation is an ongoing discussion in the Python community. If you want to know more about it,this might be an interesting article for you.
Generally, spaces are the preferred indentation method.
But whatever you choose stick with it! Especially because Python 3 doesn't allow mixing tabs and spaces for indentation.

2. Maximum Line Length

Generally, you should aim for a line length of 79 characters in your Python code. If you follow this convention:

- you can easily compare code when you open them side-by-side.
- your code is easier to read and understand because you can see the whole expression without scrolling horizontally

3. Blank Lines and Whitespaces

In Python scripts, you should surround top-level function and classes by two blank lines:

```
def sum_list(numbers):
    sum_numbers = 0
    for x in numbers:
        sum_numbers += x
    return sum_numbers


def max_num_in_list(numbers):
    maximum = numbers[0]
    for a in numbers:
        if a > maximum:
            maximum = a
    return maximum
```

However, between a bunch of related one-liners you should avoid blank lines.

Additionally, you should use whitespaces sensibly.

Do

```
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

```

```

Don't

```
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Do

```
x = 1
y = 2
long_variable = 3
```

Don't

```
x             = 1
y             = 2
long_variable = 3
```

4. Imports

When you're working with Python for data science, you will often have to import libraries.
You should always import libraries at the start of your script.
Also, several imports should usually be on separate lines:

Do:

```
import sys
import os
```

Don't

```
import sys, os
```

Additionally, you should respect a specific order when importing your libraries:

1. Standard library imports.

2. Related third-party imports.
3. Local application/library specific imports.

5. Comments

Comments start with the `#` symbol. Anything after the hashtag does not get executed by the interpreter.
Comments are used for in-code documentation in Python. They should be detailed enough such that everyone who reads the code can get a proper understanding of what you are doing and how it is being used with other pieces of the code. Yet, use them sparingly. Prefer code readability to writing a lot of comments!
Remember: Just like the rest of your code, comments should not be longer than 79 characters!

There's several types of comments:

1. You use block comments to explain code that is more complex or unfamiliar to others. They are usually a bit longer and apply to some or all of the code that follows. Block comments are indented at the same level as the code. Each line of a block comment begins with the hashtag `#` and a single space. If you need to use more than one paragraph, they should be separated by a line that contains a single `#`.

2. Inline comments should be used that often. Sometimes they might help you to remember what a specific line of code means. Use inline comments on the same line of a statement, following the code itself, they also start with `#` and a single space.

```
counter = 0  # initialize the counter
```

3. Documentation strings or docstrings is what you use at the beginning of public modules, files, classes and methods to properly describe them. These type of comments start with """ and end with """:

```
    """
    This module is intended to provide functions for scientific computing
    """
```

Especially when you write functions it is useful to explain what they do, i.e. what it takes and what it returns -This is important, not only for other people, but also for yourself, to understand your code in the future.

```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""
    if num >= 0:
        return num
    else:
        return -num
```

6. Naming Conventions

In Python, there is a set of rules for naming variables, types, functions, and other entities in your source code and documentation:

- For variables, functions, methods, packages and modules use:
  `lower_case_with_underscores`
- For classes and exceptions use:
  `CapitalizedWords`
- For protected methods and internal functions:
  `_single_leading_underscore(self, ...)`
- For private methods:
  `__double_leading_underscore(self,...)`
- Constants:
  `ALL_CAPS_WITH_UNDERSCORES`
- For "magic" objects or attributes that live in user-controlled namespaces:
  `__double_leading_and_trailing_underscore__`.
  For example, `__init__`, `__import__` or `__file__`.
  You should never invent such names, but only use them as documented.

In General:
Avoid one-letter variables (esp. l, O, I). Except for very short blocks, when the meaning is clearly visible from the immediate context:

```
for e in elements:
    e.mutate()
```

Prefer "reverse notation":

```
elements = ...
elements_active = ...
elements_defunct = ...
```

Instead of:

```
elements = ...
active_elements = ...
defunct_elements ...
```