1. Introduction

To understand code better and faster and to have consistency through the code, it's important to follow and adhere to code guidelines. Code guidelines is about consistency. Consistency in code is important. Consistency within a project is more important. Consistency within one module or function is the most important. This document provides the guidelines for Python Code at School Simplified IT Dept. – Bot Development and is intended to be a reference of compliance with those guidelines. The guidelines are also based on the PEP8 – Style Guide for Python Code and the PEP257 – Docstring Conventions and also provides internal guidelines to work with the library discord.py.

2. Code Layout

2.1. Indentation

Use 4 spaces for one indentation level. **In PyCharm, 1 tab is 4 spaces.**

2.1.1. Function definition and call

```
# Correct:

# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

```
# Wrong:

# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

2.1.2. If-Statements

For `if`-statements there are no specific rules on how to do indent. However, acceptable options include, but are not limited to:

```
# No extra indentation.
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

# No extra indentation with an empty line after if-statement.
# PREFERRED
if (this_is_one_thing and
    that_is_another_thing):

    do_something()
```

2.1.3. Closing brace/bracket/parenthesis

The closing brace/bracket/parenthesis on multiline constructs may either line up under the first non-whitespace character of the last line of list, as in:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )
```

or it may be lined up under the first character of the line that starts the multiline construct, as in:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

2.2. Line Length

We don't apply this rule, even when PEP8 requires to limit all lines to a maximum length of 79 characters. However, limit it to something reasonable. In PyCharm the limit is 120 characters by default which is marked with a thin, grey line.

IMAGE HERE

The preffered way to wrapping long lines is by using the Python's implied line continuation inside brackets. Long lines can also be broken over multiple lines by using a **backslash** for line continuation if it's needed.

```
# Python's implied line continuation inside brackets
if (True == True and False == False
    or False == False and True == True)

# Backslash line continuation
with open('/path/to/some/file/you/want/to/read') as file_1, \
     open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```