Coding Standards

Published by Calum Oke

Contents

1. Declaration

Since all programming will be done in python, following the **PEP 8 -- Style Guide for Python Code** is a recommended style guide, which this project will similarly be following.

1.1 Variables

Variables should be declared using underscore case with all characters being in lower case and have self-explanatory names. An underscore character should be placed in between the end of a word/name and beginning of a new one within the variable. Some examples are shown below:

```
foo = "Hello world"
foo_bar = [1, 2, 4, 8]
this_variable_name_is_long = 7 * 7
```

1.2 Constants

Constant variables use the same principle of underscore case, however all characters in the variable should be upper case. The purpose of this variable type is to remove “magic numbers”, which are numbers appearing to be inputted without context. Constants should also be labelled near the top of the .py file. Examples shown below:

```
MAX_VALUE = 50
MILE_TO_KILOMETRE = 1.6
PREFIX_CHARACTERS = ["bi", "tri", "un"]
```

1.3 Functions

Function names will use the same character styling as variables, as a function will be identified from the parenthesis. Names should also indicate the purpose of the function without excessive characters. Examples shown below:

```
def multiply_numbers(a, b):
    c = a * b
    return c

multiply_numbers(4, 16)
add_to_list("Milk")
result = sort_list(list_of_items)
```

1.4 Classes

Class names should be labelled using the capital word convention, with a capital character to indicate a new word/name in the class. Examples shown below:

```
class Person():
    def __init__(self, name):
        self.name = name

jill = Person("Jill")
jill.name
x = MyClass()
```

2. Spacing

Appropriate spacing important to ensure easy readability and reduce excessive whitespace usage.

2.1 Indentation

A classic rule of indentation in python is **4 space characters per indentation level**. An example is show below (*Note: indentation is how Python distinguishes each step)*:

```
for each_car in range(cars):
    for each_letter in cars:
        list_of_cars.append(car)
        print(each_letter)
print(list_of_cars)
```

When using long functions with multiple arguments, they can be laid out in multiple lines, as long as the indentation is 4 spaces after the first function name character. An example shown below:

```
def long_function(
        var_one, var_two,
        var_three, var_four,
        var_five):
    return var_one + var_two \
           + var_three + var_four \
           + var_five

print(long_function(24, 45, 56,
          72, 1))
```

2.2 Whitespace Usage

A single whitespace character should be used after every operation, comma or word/variable for easier readability. However, they should not be used after a left bracket/parenthesis and before the right one, as it adds excessive white-spacing. Examples are shown below:

```
print(multiply_nums(value_one, value_two, value_three))
list_of_items = ['a', 12, 45.12, 'abc']
x = y * z + q
```

Not:

```
print( list_of_items [0] [1] )
multiply_nums ( value_one , value_two , value_three )
x=y*z+q
```

However, exceptions of not using whitespace characters around the '=' sign are applied when assigning values to parameters or in-built functions that can have inputted values. Examples below:

```
def deduct(value_a, value_b=1.1):
    return addition(value_a*20, value_b)
```

2.3 Line Length

The maximum line length should not go over 80 columns wide. Long lines of code should be broken over multiple lines by using the '\' character (except in definitions) to indicate code continues over a new line. Examples below:

```
def lots_of_input_values(value_one, value_two, value_three, value_four,
                         value_five, value_six):
    return value_one + value_two + value_three * value_four - value_five \
           + value_five
```

To improve readability, it is preferred that related code is layered out by one value per line. Example below:

```
def lots_of_input_values(value_one,
                         value_two,
                         value_three,
                         value_four,
                         value_five,
                         value_six):
    return value_one \
           + value_two \
           + value_three \
           * value_four \
           - value_five \
           + value_five
```

Note that the arithmetic operator is in front of every value on the new line. This is how the related code should be implemented as well. *Example of return values may be a bit excessive in new lines, however it is to show how they should be laid out on each line*.

2.4 Line Separation

Lines of white space should be implemented between each unrelated block of code or blocks that have different functionality. Examples below:

```
from math import *
from tkinter import *

MAX_VALUE = 50
SIDES_OF_DICE = 6

abc = 'Welcome to this sample'
x = 5
list_of_items = ["abc", 1, 2.5]

for item in list_of_items:
    print(item)

empty_list = []
counter = 0

while True:
    counter = counter + 1
```

Imports, constants, variables etc. are separated by a newline to indicate new functionality

3. Commenting

3.1 Inline

Inline commenting should be try to be avoided, unless code is not interoperable by a person or requires explanation of purpose. For example:

```
while counter < height:
    x = x + 1 # line for right border
```

Usually using self-explanatory variable names or block comments eliminates the usage of inline comments

3.2 Block

Block commenting should always be used when complex blocks of code occur that need functionality explaining. These comments should always be placed above each block of code and under each line separation. Examples shown below:

```
shoes_list = []

# Adds all shoes to the empty list from the shoe webpage via regex
for shoe in range(html_shoe_page):
    print(shoe)
    shoes_list.append(shoe)
```

4. Code Layout

A general Python document should be laid out in the same concept as below:

```
from math import *
from tkinter import *
from timeit import Timer

# Constants
KM_CONVERT = 1.6
SIDES_OF_DICE = 6
MAX_VALUE = 10000

# Variables
cars_list = ['Sedan', 'Hatchback', 'Coupe']
x = 21

def convert_to_km(mile):
    return mile * KM_CONVERT

def multiply_nums(value_one, value_two, value_three):
    return (value_one * value_two * value_three)

# Loops to multiply the highest number before reaching the maximum value
while result < MAX_VALUE:
    result = result + multiply_nums(10, 3, 21)
    print(result)

window = Tk()
window.mainloop()
```

This example displays briefly how the code should be laid out. Imports should always be placed at the very top of a program. Constants and global variables should be separated and placed near the top. Function definitions should be placed under variable declarations and above main program functionality. Appropriate commenting structure is also shown above.

References

van Rossum, G., Warsaw, B., & Coghlan, N. (2001, July 5). *PEP 8 -- Style Guide for Python Code*. Retrieved from python.org: https://www.python.org/dev/peps/pep-0008/#programming-recommendations