Anurag Tiwari, FYIT-90

Date
Page

IT Tools and Practices

* PEP in Python *

(Q.1) What is PEP?

Ans The PEP is an abbreviation form of Python Enterprise proposal writing code with proper logic is a key factor of programming but many other important factors can affect the code quality. The developer's coding style makes the code much reliable, and every developer should keep in mind that python strictly follows the way of order and format of the string. Adaptive a nice coding style makes the code more readable. The code becomes easy for end-user.

PEP 8 is a document that provides various guidelines to write the readable in Python. PEP 8 describes how the developer written in 2001 by Guido Van Rossum Barry Warsaw, and Nick coghlan. The main aim of PEP is to enhance the readability and consistency of code.

(Q.2) Why PEP 8 is important?

Ans PEP 8 enhance the readability of the python code, but Why is readability so important? Creator of python, Guido Van Rossum said, "code is much more often than it is written". The code can be written in a few minute, a few hours, or a whole day but once we have

written the code, we will never rewrite it again. But sometimes, we need to reuse the code again and again.

At this point, we must have an idea of why we wrote the particular line in the code. The code should reflect the meaning of each line. That's why readability is so much important

- Naming Convention:

When we write the code, we need to assign name to many things such a variables, functions, classes, packages, and alot more things. Selecting a proper name ~~back recall what~~ will save time and energy. When we look back to the life after sometime, we can easily recall what a certain variable, function or class represents. Developers should avoid choosing inappropriate names. The naming convention in Python is slightly messy, but there are certain convention that we can follow easily.

Examples: Single Lowercase letter

```
a=10
```

Single uppercase letter

```
A=10
```

Lowercase

```
var=10
```

Lower_case_with_underscores

```
number_of_apple=5
```

Anurag Tiwari, FYIT-90

Date
Page

~~Upper~~

UPPERCASE
VAR = 6

UPPER_CASE_WITH_UNDERSCORES
NUM_OF_CARS = 20

CapitalizedWords (or camelcase)
Number of BOOKS = 100

* Name style:

Type	Naming Convention	Examples
Function	We should use the lowercase words or seperates words by the underscor	my function, my_function.
Variable	We should use a lowercase ~~We~~ letter, words, or seperate words to enhance the readability	a, var Variable_name
Class	The first letter of class name should be capitalized: use camel case. Do not seperate words with the underscore.	Myclass, Form, Model.
Method	We should use a lowercase letter, words, or seperate words to enhance readability	class_method method.

Date
Page

~~Method~~ Constant	We should use a ~~lowercase~~ short, uppercase letters, words or seperate words to enhance the readability.	MYCONSTANT, CONSTANT, MY_CONSTANT
Module	We should use a lowercase letter, words or seperate words to enhance the readability	Module_name.py, module.py.
Package	We should use a lowercase letter, words, or seperate words to enhance the readability. Do not seperate words with the underscore.	Package, mypackage

* Code Layout:
The code layout defines how much the code is readable. In this section, we will learn how to use whitespace to impress code readability.

* Indentation:
Unlike other programming languages, the indentation is used to define the code block in Python. The indentations are the important part of the python programming language and it determines the level of lines of code. Generally, we use the 4space for indentation.

Example:
```
x=5
if x==5:
    print('x is a larger than 5')
```

Date
Page

In above example, the indented print statement will get executed if the ~~co~~ condition of if statement is true. This indentation defines the code block and tells us what statements execute when a function is called or conditi trigger.

- Tabs vs space
We can also use the ~~tabbte~~, tabs to proviole the consecutive space to indicate the indentation. but whitespaces are the most preferable python 2 allows the mixing of ~~table~~ tabs and spaces but we will get an error in python.

* Indentation following Line Break.
It is essential to use indentation when using continuations to keep the lines to fewer than 79 characters. It provides the flexibility to determining between two lines of code and a single line of code that extends two lines.

Example: ① # correct way:
# Aligned with opening delimiter.
obj = func_name (argument_one, argument_two,
argument_three, argument_four)

② #first line doesn't has any argument.
We add 4 spaces from the second line to descriminate arguments from the rest.
def function_name (
argument_one, argument_two, argument_

```
three, argument_four):
print (argument_two)

# 4 space indentation to add a level
foo = long_function_name(
var_one, var_two,
var_three, var_four)
```

* Use docstring
Python provides the two types of document strings or docstring - single line and multiple lines. We use the triple quotes to define a single line or multiline quotes. So, these are used to describe the function or particular program.

eg:-
```
def add(a,b):
    """ This is simple method """
    """ This is
    a
    Simple add program to add
    the two numbers """
```

* Should a Line Break Before or after a Binary Operator?
The lines break or after a binary operation is a traditional approch. But it affects the readability extensively because the operators are scattered across the different screens, and each operator is kept away from its operand and onto the previous line.

Date
Page

eg:① #Wrong
# operators lit far away from their operand
marks = (english_marks + math_marks +
science_marks + biology_marks + physics_
marks)

eg: ② #correct:
# easy to match operators with operands
Total_mark = (English_marks + math_marks +
science_marks biology_ marks + physics_marks)

Python allows us to breack line before or after a binary operator as long as the convention is consistent locally.

* Importing module

- We should import the modules in the separate line as follows:

```
import pygame
import os
import sys
```

- Wrong

```
import sys, os ....
```

- We can also use the following approach

```
from subprocess import popen, PIPE
```

Date
Page

- The import statement should be written at the top of the File or just after any module comment. Absolute imports are the recommended because they are more readable and tend to be better behaved.

```
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example
```

However, we can use the explicit relative imports instead of absolute imports, especially dealing with complex packages.

* Blank Lines

Blank Lines can be improved the readability of python code. If many lines of code branched together the code will become harder to read. We can remove this by using the many blank vertical lines, and the reader might need to scroll more than necessary. Follow the below instructions to add vertical whitespace.

- Top-Level function and classes with two lines. Put the external vertical space around them so that it can be understandable.

⇒ Class First class:

```
pass
```

Date
Page

```
class secondclass:
    pass

def main_function():

    return None
```

- Second blank line inside classes - The function that we define in the class is related to one another.

Eg:-
```
class Firstclass:
    def method_one (self):
    return None

    def second_two (self):
        return None.
```

- Use blank lines inside the function - Sometimes, we need to write a complicated function has consists of several steps before the return statement. So we can add the blank line between each step

eg:-
```
def cal_variance (n_list):
list_sum = 0
for n is n_list:
    list_sum = list_sum + n
    mean = list_sum / len(n_list)
```

Date
Page

```
square_sum = 0
for n in n_list:
    square_sum = square_sum + n**2
mean_squares = square_sum / len(n_list)
return mean_square - mean**2
```

- The above way can remove the whitespaces to improve the readability the code.

* Put the closing Braces
We can break lines inside parentheses, brackets using the Line continuations, PEP 8 allows us to use closing braces in implies line continuations.

- Line up the closing brace with the first non-whitespace

```
list_numbers = [5, 4, 1, 4, 6, 3, 7, 8, 9]
```

- Line up the closing braces with the first character of line

```
list_numbers = [5, 4, 1, 4, 6, 3, 7, 8, 9]
```

Both method are suitable to use, but consistency is key. So choose any one and continue with it.

* Comments
Comments are the integral part of the any programming language. These are the

Date
Page

belt way to explain the code. When we documented our code with the proper comments anyone can able to understand the code but we should remember the following points

- start with the capital letter, and write complete sentence.
- Update the comment in case of a change in ~~code~~ code.
- Limit the line length of comments and docstrings to 72 characters.

* Block comment

Block comments are the goode choice for the small section of code. Such comments are useful when we write several line codes to perform a single action such as ~~ite~~ iterating a loop. They help us to understand the purpose of the code.

PEP8 provides the following ~~res~~ rules to write comment block

- Indent block comment should be at the same level
- Start each line with the # followed by a single space.
- Seperate line using the single #

Date
Page

eg:
```
for i in range (0,5):
# loop will iterate over five times
# new line character
print (1, '\n')
```

We can use more than paragraph for the technical code.

* Inline comments

Inline comments are used to explain the single statement in a piece of code. We can quickly get idea of why we wrote that particular line of code. PEP 8 specifies the following rules for the inline coments

- start comments with # and single space
- Use inline comments carefully.
- We should separate the inline comment on the same line as the statement they refer.

eg:
```
a=10 # The a is variable that holds integer values
```

Sometimes, we can use the naming convention to replace the inline comment.

eg:
```
x= 'Peter Decosta'
# This is a student name
```

We can use the following naming convention

eg:-
```
student_name = 'Peter Decosta'
```

Date
Page

Inline comments are essential but block comments make the code more readable.

* Avoid Unnecessary Adding Whitespace
In some cases, use of whitespace can make the code much harder to read. Too much whitespaces can make code overly sparse and difficult to understand. We should avoid adding whitespaces at the end of a line. This is known as trailing whitespaces

eg: ①
```
# Recommended
List1 = [1, 2, 3]

# Not Recommeded
List 1=[1, 2, 3]
```

eg: ②
```
x = 5
y = 6

# Recommended
print(x, y)

#Not recommended
print(x, y)
```

* Programming Recommendation
As we know that, there are several method to perform similar tasks in Python. In this section. we will see some of the suggestions of PEP 8 to improve the consistency.

Date
Page

Avoid comparing Boolean Values using the equivalence operator.

eg:
```
# Not recommended
bool_value = 10>5
if bool_value == True:
  return '10 is bigger than 5'
```

we shouldn't use the equivalence operator == to compare the Boolean values. It can only take the True and False.

eg:
```
# Reccomended
if my_bool:
  return '10 is bigger than 5'
```

* Empty sequences are false in if statements. If we want to check whether a given list is empty, we might need to check the length of list, so we need to avoid.

Eg:
```
# Not recommended
list1 = []
if not len(list1):
  print('List is empty!')
```

However, if is any empty list, set or tuple.

Eg:
```
Recommended
List1 = []
if not list1:
  print ('List is empty!')
```

Date
Page

The second method is more appropriate; that's why PEP 8 encourages it.

* Don't use not is in if statement
There are two options to check whether a variable has a defined value. The first option is with x is not None, as in the example.

Eg:
```
# Recommended
if x is not Non:
  return 'x exists!'
```

A second portion is to evaluate x is None and if statement based on not the outcome

Eg:
```
# Not Recommended
if not x is none:
  retorn 'x exists!'
```

* Conclusion.

We have discussed the PEP 8 guidelines to make the code remove ambiguity and enhance readability. These guideline improve the code, especially when sharing the code with potential employees or collaborators.