

Name: Jadhav Dushna Hameerbhai  
Class:- FYIT Roll No: 32 Div: A

### ASSIGNMENT

#### \* PEP 8 in Python

##### \* What is PEP?

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's 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 formatting of the string.

Adopting a nice coding style makes the code more readable. The code becomes easy for end-users.

PEP 8 is a nice document that provides various guidelines to write the readable in Python. PEP 8 describes how the developer can write beautiful code. It was officially written in 2011 by Guido van Rossum, Barry Warsaw, and Nick Coghlan. The main aim of PEP is to enhance the readability and consistency of code.

## \* Why PEP 8 is important?

PEP 8 enhances the readability of the Python code, but why is readability so important? Let's understand this concept.

Creator of Python, Guido van Rossum said, "Code is read more often than it is written." The code can be written a few minutes, 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 read 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.

We will describe few important guidelines for writing effective code that can be ready by others as well.

## \* Naming Convention.

When we write the code, we need to assign name to many things such as variables, functions, classes, packages, and a lot more things. Selecting a proper name will save time and energy. When we look back to the file 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 conventions that we can follow easily. Let's see the following naming conventions.

Example -

Single lowercase letter.  
 $a = 10$

Single uppercase letter.  
 $A = 90$

Lowercase

$var = 10$

Lower case with underscores.  
 $number\_of\_apple = 5$

UPPERCASE

$VAR = 6$

UPPER\_CASE\_WITH\_UNDERSCORES

$NUM\_OF\_CARS = 20$

CapitalizedWords (or CamelCase)

$NumberOfBooks = 100$

\* Name Style.

Below is the table that specifies some of the common naming styles in Python. Consider the following table.

## Type

## Naming Convention

Example

### Function

We should use the lowercase words or separates words by the underscore.

myfunction  
my function

### Variables

We should use a lowercase letter, words, or separate words to enhance the readability.

### Class

The first letter of class name should be capitalized; use camel case. Do not separate words with the underscore.

MyClass, John

### Method

We should use a lowercase letter, words class\_method, or separate words to enhance readability.

method.

### Constant

We should use a short, uppercase letter, MYCONSTANT, words, or separate words to enhance the readability.

MY\_CONSTANT

### Module

We should use a lowercase letter, words, or separate words to enhance the readability.

Module name: module.py

### Package

We should use a lowercase letter, words, or separate words to enhance the readability. Do not separate words with the underscore.

package, mypack

## \* Code Layout:

The code layout defines how much the code is readable. In this section, we will learn how to use whitespace to improve 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 4 space for indentation. Let's understand the following example.

Example:-

```
x = 5
```

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

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

### • Tabs vs. Space.

We can also use the tabs to provide the consecutive spaces to indicate the indentation, but whitespaces are the most preferable. Python 2 allows the mixing of tabs and spaces but we will get an error in Python 3.

Name: Jadhav Darshna Harishbhai  
Class: F4IT Roll No: 32

Rainbow  
PAGE:  
DATE:

## Indentation following Line Break.

It is essential to use indentation when using line continuations to keep the line to fewer than 79 characters. It provides the flexibility to determine between two lines of code and a single line of code that extends two lines. Let's understand the following example.

Example:-

# Correct Way:

# Aligned with opening delimiter.

obj = func\_name (argument\_one, argument\_two,  
argument\_three, argument\_four)

We can use the following structure.

# first line doesn't have any arguments.

# we add 4 spaces from the second line to discriminate arguments from the rest.

def function\_name (

    argument\_one, argument\_two, argument\_three, a  
    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 documents strings or docstring - single line and multiple lines. We use the triple quotes to define a single line or multiple / multiline quotes. Basically, these are used to describe the function or particular program. Let's understand the following example.

Example -

```
def add (a,b):  
    """This is simple add 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 before or after a binary operation is a traditional approach. But it affects the readability extensively because the operators are scattered across the different lines, and each operator is kept away from its operand and onto the previous lines. Let's understand the following example.

Example:-

#Wrong:

```
#operators sit far away from their operands  
marks = (english_marks + math_marks +  
         science_marks + biology_marks) +  
         physics_marks .
```

Name: Jadhav Danhna Harshottar  
Class: F4 IT ROLL NO: 32

Rainbow  
PAGE:  
DATE:

Python allows us to break line.  
As we can see in the above example, it seems quite messy to read. We can solve such types of problems by using the following structure.

Example -

# coded

# easy to match operators with operands

Total marks = English\_marks +

+ math\_marks

+ (science\_marks + biology\_marks)

+ physii\_marks.

Python allows us to break 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

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 bunched 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 extra vertical space around them so that it can be understandable.

```
class FirstClass:  
    pass
```

```
class SecondClass:  
    pass
```

```
def main_function():  
    return None
```

Name: Jadhav Darshna Harishbhai  
Class: FYIT Roll No: 32

Rainbow  
PAGE:  
DATE:

- Single blank line inside classes- The functions that we define in the class is related to one another. Let's see the following example -

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. Let's understand the following example

```
def cal_variance(n_list):
```

```
    list_sum = 0
```

```
    for n in n_list:
```

```
        list_sum = list_sum + n
```

```
    mean = list_sum / len(n_list)
```

```
    square_sum = 0
```

```
    for n in n_list:
```

```
        square_sum = square_sum + n**2
```

```
    mean_square = square_sum / len(n_list)
```

```
    return mean_square - mean**2
```

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

## \* Put the closing Braces

We can break lines inside parentheses, brackets using the Lines continuations. PEP 8 allows us to use closing braces in implicit line continuations. Let's understand the following example.

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

list\_number = [

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 methods 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 best 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 letters, and write complete sentence.
- Update the comment in case of a change in code.
- Limit the line length of comments and documentation to 72 characters.

## \* Block Comments.

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

PEP 8 provides the following rules to write comment block.

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

Let's see the following code:

for i in range (0,5):

# Loop will iterate over i five times and print out the values of i

# new line character

print (i, '\n')

We can use more than paragraph for the technical code.  
Let's understand the following example.

#### \* Inline Comments

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

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

Following is the example of inline comments.

a=10 # The a is variable that holds integer value.

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

`x = 'Peter Deosta' # This is a student name.`

We can use the following naming convention.

Student\_name = "Peter Deosta".

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

#### \* Avoid unnecessary Adding Whitespaces.

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

Let's see the following example.

Example 1 -

# Recommended

`list1 = [1, 2, 3]`

# Not Recommended

`list1 = [1, 2, 3,`

Example 3 -

`x = 5 , y = 6`

# Recommended

`print(x, y)`

# Not Recommended

`print(x, y)`

## \* Programming Recommendation.

As we know that, there are several methods to perform similar tasks in Python. In this section, we will some more of the suggestions of PEP 8 to improve the consistency.

### \* Avoid comparing Boolean values using the equivalence operator.

```
# 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` or `False`. Let's see following example.

```
# Recommended
```

```
if my_bool:
    return '10 is bigger than 5.'
```

This approach is simple that's why PEP 8 encourages it.

\* 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 use the following approach.

# Not recommended

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

However, if there is any empty list, set, or tuple, we use the following way to check:

# Recommended

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

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 following example.

### # Recommended

if  $x$  is not None:  
    return ' $x$  exists!'

A second option is to evaluate  $x$  is None and if statement based on not the outcome.

### # Not recommended

if not  $x$  is None:  
    return ' $x$  exists!'

Both options are correct but the first one is simple, so PEP8 encourages it.

## \* Conclusion :

We have discussed the PEP8 guidelines to make the code remove ambiguity and enhance readability. These guidelines improve the code, especially when sharing the code with potential employers or collaborators. We have discussed what PEP8 is and why it uses, how to write code that is PEP8 compliant. Moreover, we have a brief introduction to the naming conventions. If you want more information regarding the PEP8, ~~you need~~