Maitreyi Badkar
FYIT, Roll no: 4.

What is PEP?

The pep is an abbrivation form of python entrprise proposal writing code with proper logic is a key factor of programming, but many other important factors can affect the code quality. The developers coding style makes the code much reliable, & every developer should keep in mind that python strictly follows the way of order & format of the string adapting a nice coding style makes the code more readable. The code becomes easy for end user.

Pep 8 is a doument that provides various guidelines to write the readable in python. PEP 8 describe how the developer can write beautiful code. It was offically written in ~~2021~~ 2001 by Guido van Rossum, Barry warsaw & nick coghlan. The main aim of PEP is to enhance the readability & consistency of code.

Why PEP 8 is important?
PEP 8 enhances the readability of python code, but why is readability so important? let's understand this concept

creator of python, Guido van Rossum said, "code is much more often than it written." The code can be written in a few minutes, few hours, or a whole day but once we have written the code, we will never rewrite it again.

Roll no:4

But sometimes we need to read the code again & again

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

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

Naming conventions:-

When we write the code, we need to assign name to many things such as variables, functions, classes, packages & a lot more things. Selecting a proper name will save time & energy. When we look back to the file after sometime, we can easily recall what a certain variable, function or a class represent. Developers should avoid choosing inappropriate names.

The naming convention in python is slightly messy, but there are certain conventions that we can follow easily. Lets see the following naming conventions.

Roll no:4

example :-

single lowercase letter
a = 10

single uppercase letter
A = 10

lowercase
var = 10

lower_case_with_underscores
number_of_apples = 5

uppercase
VAR = 6

UPPERCASE_WITH_UNDERSCORE
NUM_OF_CARS = 20

Capatalizedwords
Numberofbooks = 100

Roll no:4

Name style

Type	Naming conventions	examples.
functions	We should use the lower-case words or seperate words by underscores	myfunction my-function
Variable	we should use a lowercase letter, words, or seperate words to enhance the readability	a, var, variable_name
class	The first letter of class name should be capatalized. do not seperate words with underscore	myclass, Form, model.
Method	We should use a lowercase letter, words, or seperate words to enhance readability	class_method, method
constant	We should use a short uppercase letter, words or seperate words to enhance readability	MYCONSTANT, CONSTANT, MY_CONSTANT

Roll no: 4

module	we should use a lower case letter, words. or seperate words to enhance readability	module_name.py, module.py
Package	we should use lowercase letters, words or seperate words to enhance the readability. Do not seperate words with underscore	package. mypackage

Above are some common naming conventions that are useful to beautify the python code. for additional improvement, we should choose the name carefully.

Code Layout.

The code layout defines how much code is readable. In this section, we will learn how to use whitespace to improve code readability

Identation

unlike other programming languages. the identation is used to define the code block in python. The identation are the important part of the python programming language & it determines the level of lines of code. generally we use the 4 space for idntation.

Roll no:4

Let understand the following eg:-

eg:-

```
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 identation defines the code block & tells us what statements executes when a function is called or condition triggered

* Tabs vs space

We can also use the tabs to provide the consequtive spaces to indicate the identation, but whitespace are most preferable. python 2 allows the mixing of tabs & spaces. But we will get an error in python 3.

Identation following line break.

It is essential to use identation when using line continuation to keep the lines to fewer than 79 characters. It provides the flexability to determining between two lines of code & a single line of code that extends two lines.

Roll no:4

~~[illegible]~~ Lets understand the following 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 doesnt has any argument
we add 4 spaces from the second line to discriminate
arguments from the rest

```
def function_name (
    argument_one, argument_two, argument_three, argument_four):
  print (argument_two)
```

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

Roll no:4

use docstring:-

python provides 2 types of document strings or docstring single line & multiple lines. we use triple line quotes to define a single line or multiple line quotes. Basically these are used to describe the function or particular program. Let understand the following. Let understand the following examples.

Examples:

```
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 & after a binary operation ?

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 screens & each operator is kept away from its operand & onto the previous line. Let understand the following example.

Roll no: 4

example:

#wrong:

#operators sit far away from their operands marks =
(english_marks + maths_marks + (science_marks - biology_marks) +
physics_marks

as we can see in the above example, it seems quite messy
to read. we can solve such types of problems using
following structure

example :-

#correct:

#easy to match operators with operands

```
Total_marks = (english_marks
              + math_marks
              + (science_marks - biology_marks)
              + physics_marks
```

Roll no:4

python allows us to break line before or after a binary operator, as long as the convention is consistently locally

importing ~~total~~ module:

we should import the modules in the seperate lines 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 file or just after any module comment. Absolute imports are recommended because they are more readable & tend to be better behaved.

Roll no: 4

```
import mypkg.sibling
from ~~p~~ mypkg import sibling
from mypkg.sibling import example
```

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

Blank lines:-

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

Top level function & classes with 2 lines:-

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

```
class firstclass:
 pass


class secondclass:
pass.
```

12

Roll no: 4

```
def main_function():

return none
```

- single blank line inside classes:
The functions that we define in the class is related to one another.

Let 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 consisting of several steps before return statement. so we can add the blank line between each step.

Let understand the following eg:-

```
def cal_variance (n_list):
  list_sum=0
```

Roll no: 4

```
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_squares = square_sum / len (n_list)

return mean_squares - mean**2.
```

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

Put closing braces.

we can break lines inside parenthesis brackets using line continuations. PEP 8 allows us to use closing ~~bracket~~ braces in implies lines continuations.

Let understand this example.

• line up the closing brace with the first non-whitespace character.

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

Roll no: 4

• Line up the closing brace 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 to choose any one & continue with it

* Comments :

Comments are the integral part of any programming language. These are the best way to explain the code. When we documented our code with proper comments anyone can be able to understand the code. But we should remember the following points.

• Start with the capital letter, & write complete sentences

• Update the comment in case of change of code.

• Limit the line length of comments & docstring to 72 characters.

Roll no: 4

Block comment :-

Block comment are the good choice for the small section of code. Such comment are useful when we write several line code 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 # followed by a single space

Separate line using #.

Lets see the following code

```
for i in range (0,5):

    # Loop will iterate over i five times & print out the value of i.

# new line character

print (i,'\n').
```

Roll no:4

we can use more than paragraph for the technical code.

Let understand the following code

Inline comment
inline comment 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 rule for inline comment

- start comment with the # & single space
- use inline comment carefully
- we should separate the inline comment on the same line as the statement they refer

following is the example

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

sometimes, we can use the naming convention to replace the inline comment

x = "Peter Decosta" # This is a student name.

Roll no:4

we can use the following naming convention

```
student_name = 'Peter Decosta'
```

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 & difficult to understand. we should avoid adding whitespaces at the end of a line. This is known as trailing whitespaces.

Let see examples:-

eg:-

```
# Recommended
list1 = [1, 2, 3]

# Not recommended
list1 = [1, 2, 3]
```

Roll no:4

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 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'
```

Roll no: 4

we should'nt use the equivalence operator == to compare the Boolean values. It can only take the true or false.

see the example.

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

The approach is simple that why PEP 8 encourage it

Empty sequences are false in if statements

if we want to check whether given list is empty, we might need to check the length of list, so we need to avoid the following approach

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

However if there is any empty list, set, or tupple we can use the following way to check.

Roll no: 4

```
# Recommended
list1 = []
if not list1:

    print ('List is empty!')
```

This method is more appropriate that why PEP8 encourages it.

Dont use not is in if statement
There are two options to check whether a variable has a defined value. The first option with x is not none

```
# Recommended
if x is not none:
    return 'x exists!'
```

second option to evaluate x is none.

```
# Not Recommended
if not x is None:
    return 'x exists!'
```

Both options are correct but the first one is simple.