Page No.:
Date: / /

IT Tools & Practices

Name: Riddhi · Hitesh · Kanabar
Class: FYIT Roll NO: 36.

* PEP in Python *

① 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 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 can write beautiful code. It was officially 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.

② Why PEP 8 is important?

→ PEP 8 enhances 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 minutes, a

Name : Riddhi . Hitesh . Kanabar
Class : FYIT Roll NO : 36

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 ~~WAt~~ 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 as variables, functions , classes, packages , and alot more things . Selecting a proper name will save time and energy . When we look back to the file after sometime , we can easily recall what a certain variable , function or 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 .

Examples : Single lowercase letter .
a = 10

Single uppercase letter
A = 10

Name : Riddhi · Hitesh · Kanabar
Class : FYIT Roll NO : 36

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 :

Type	Naming convention	Example
Function	We should use the lowercase words or separates words by the underscore	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 capitalized ; use camel case. Do not seperate	Myclass, Form, Model.

Name : Riddhi . Hitesh . Kanabar
Class : FYIT Roll NO : 36

	words with the underscore.	
Method	We should use a lowercase letter, words, or seperate words to enhance readability.	class_method, method.
Constant	We should use a short, uppercase letters. words or seperate words to eh enhance the readability.	MY CONSTANT, 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 separate 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 improve code readability.

Page No.:
Date: / /

Name : Riddhi .Hitesh .Kanabar
Class : FYIT Roll NO : 36

* 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.

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

In 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 trigger.

- 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 : Riddhi Hitesh Kanabar.
Class : FYIT Roll NO : 36.

* Indentation following line Break.
It is essential to use indentation when using continuations to keep the line 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 discriminate 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)

Page No.:
Date. / /

Name : Riddhi . Hitesh . Kanabar
Class : FYIT ROLL NO: 36

* 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 approach. But it affects the read-ability extensively because the operators are scattered across the different screens, and each operator is kept away from its operand and onto the previous line.

Name : Riddhi .H. Kanabar.
class : FYIT Roll NO: 36

Eg : ① # Wrong.
# operators sit far away from their operands
marks = (english_marks + math_marks +
science_marks + - biology_marks + physics_marks

Eg : ② # correct :
# easy to match operators with operands.
Total_marks = (english_marks + math_marks +
science_marks - biology_marks + physics_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...
```

Page No.:
Date: / /

Name : Riddhi . Hitesh . Kanabar
Class : FYIT Roll NO: 36.

- 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 absolutes 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 white-space.

Name : Riddhi . Hitesh . Kanabar .
Class : FYIT Roll NO: 36 .

- 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

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

Eg: ① class Firstclass:
    def method_one(self):
        return None.

    def second_two(self):
        return None.

Name : Riddhi Hitesh Kanabar
Class : FYIT Roll NO : 36

- 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 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_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.

Name : Riddhi · Hitesh · Kanabar
Class : FYIT Roll NO : 36

- 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 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 letter, and write complete sentence.

- Update the comment in case of a change in code.

Page No.:
Date. / /

Name : Riddhi · Hitesh · Kanabar.
Class : FYIT Roll NO : 36.

- Limit the line length of comments and docstrings to 72 characters.

* Block comment.
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~~ 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 #.

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

We can use more than paragraph for the technical code.

Name: Riddhi .Hitesh. Kanabar.
class: FYIT ROLL NO: 36.

* 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. PEP8 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 refer.

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

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'.

Name: Riddhi · Hitesh · Kanabar
class: FYIT Roll No: 36.

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 sho-uld avoid adding whitespaces at the end of a line. This is known as trailing whitespaces.

Eg: ① # Recommended.
list 1 = [1, 2, 3]

Not Recommended.
list 1 = [1, 2, 3]

Eg: ②
x = 5
y = 6

Recommended
print (x, y)

Not recommended.
print (x, y)

Name: Riddhi Hitesh Kanabar.
Class: FYIT Roll No: 36.

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

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 or False.

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

Name: Riddhi Hitesh Kanabar
Class: FYIT Roll No: 36

* 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 there is any empty list, set or tuple.

Eg:
```
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 example.

Page No.:
Date. / /

Name: Riddhi . Hitesh . Kanabar
Class: FYIT Roll NO: 36.

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

2 second portion is to evaluate x is None and if statement based on not the outcom

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

* Conclusion

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